2000年4月9日発行　DOS/Vmagazine特別編集

SOFTBANK MOOK

2000春号 Oh!X

特集 Game Creation!

3DネットワークゲームGALAXY-NIGHTSの制作（前編）
3DシューティングゲームSION2K/VBによるスプライトライブラリ

特別企画 FlexATXによるオリジナルPC組み立て

GeneralMIDI-2の概要/いまどきのアマチュアハード製作入門
NetBSD X68Kのススメ

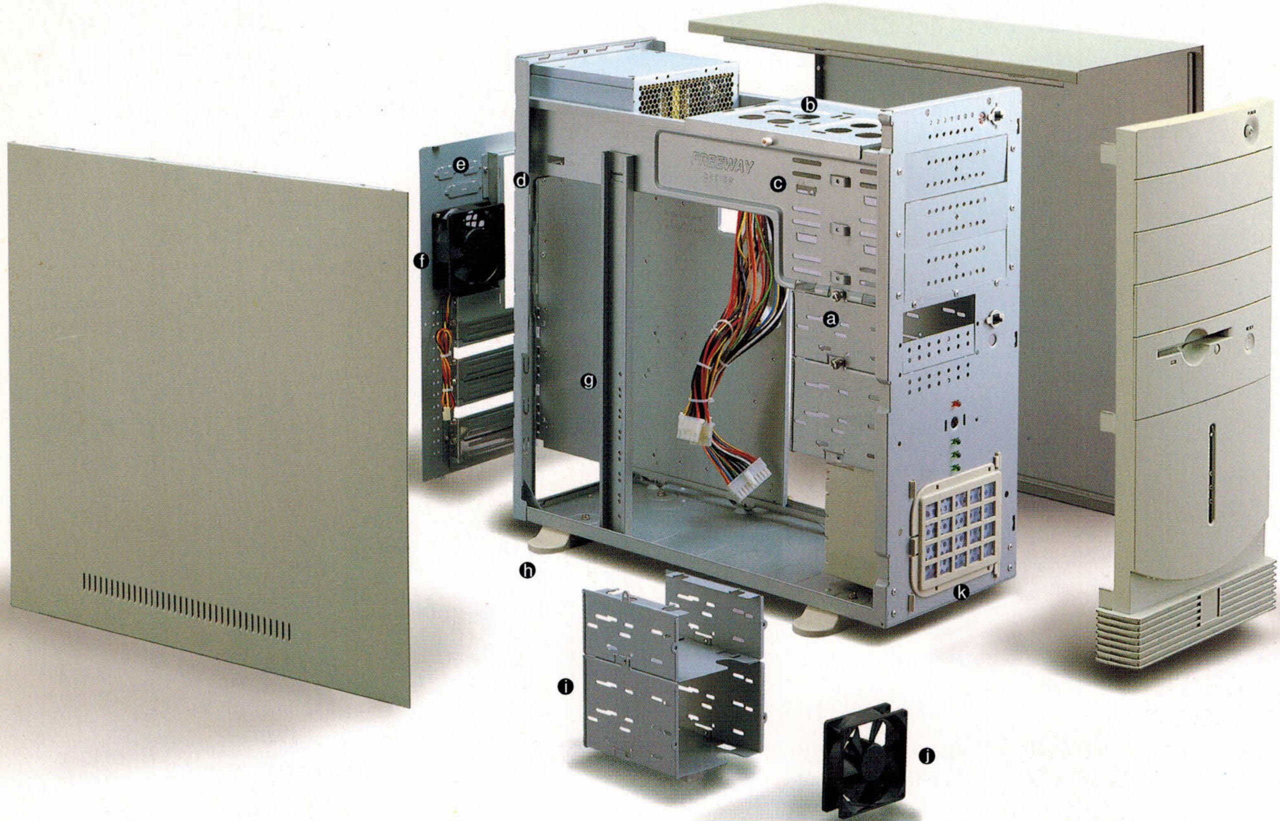
特別付録 CD-ROM2枚組

SOFT BANK Publishing
オー！エックス
2000春号
定価2400円



WAY'S DNA

より快適なコンピュータ環境を構築するために、新たな方向性を探り続けてきた FREEWAY。
そのひとつの回答が、ここにあります。
人の意志を確実にマシンへと引き継ぐ、オリジナル商品の数々。
私たちのDNAは、未来をバージョンアップする。



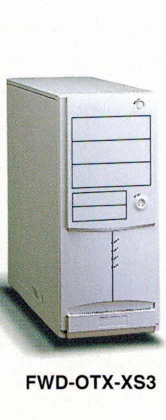
メンテナンス性・拡張性抜群 NEW DESIGNED in JAPAN
FREEWAY オリジナルPCケース

- ⓐドライバ不要のハンドスクリュー
- ⓑトップトレイにも3.5インチHDD2台取付可
- ⓒワンタッチ取付可能5インチベイ用アタッチメント
- ⓓ3種類のI/Oガスケット添付
- ⓔメンテナンスしやすいドロアタイプ
- ⓕ静音型リアファン標準装備
- ⓖ拡張カード固定用スタビライザ（別売）
- ⓗ360°回転式フットスタンド
- ⓘ取り外し可能3.5インチベイ
- ⓙ静音型フロントファン標準装備
- ⓚ洗浄可能なエアフィルター標準装備

- ○ サイドパネル、ドロアはすべてドライバー無しで開閉可能
- ○ ATX電源ユニットQuiet&Cool標準装備
- ○ 手を傷つけないエッジ加工
- ○ 3.5インチシャドーベイ5個



FWD-MAS-XS3



FWD-OTX-XS3

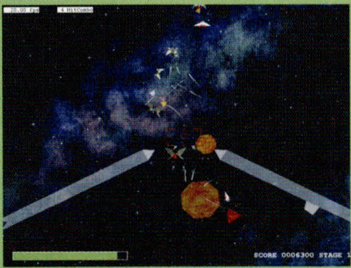


FWD-YCC-XS3



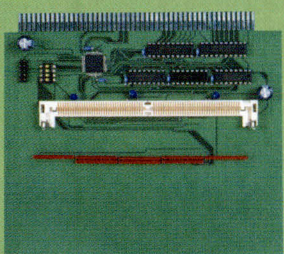
	FWD-MAS-XS3	FWD-OTX-XS3	FWD-YCC-XS3
拡張ベイ	3.5"×2（5） 5"×3	3.5"×2（5） 5"×3	3.5"×2（5） 5"×3
サイズ（H×W×L）	454×195×501.5	454×195×501.5	454×195×501.5
電　源	250W	250W	250W
希望小売価格	¥14,800	¥14,800	¥14,800

※300W電源搭載モデルもご用意しています。
※ケースサイズに突起部・フットスタンドは含まれておりません。


特集　SION2000


特別企画　FlexATX によるオリジナルPC


Direct3D IM を使う


いまどきのアマチュアハードウェア製作入門


My Memmories Your Memories


EverQuest

SOFTBANK MOOK

2000春号

Oh!X

C O N T

●特集　Game Creation!

●Step to the Black Arts

●特別企画　西暦2000年のMIDI環境

●特別企画



©2000 Yoshimizu Kei

E N T S

<スタッフ>
●編集／植木章夫
●協力／有田隆也 石上達也 市川幹人 古村 聡 須賀 真 田中順子 中森 章 広井 誠 福嶋章太 古旗一浩 松尾直樹 御木徳高 水野貴明 箕浦 誠 吉田賢司 サイバーヘッド 満開製作所
●カメラ／新井邦彦 ●校正／フィールドアップ
●イラスト／岡村直也 高橋哲史 福原 徹 山田晴久 由水 桂 シブミ
●編集長／来島 樹
●編集人／稲葉俊郎
●発行人／岡崎 眞
●DTPデザイン／クニメディア ●印刷／クニメデイア サンエー印刷

GAME

ゲームをテーマに，未来と過去を感じさせるイメージを表現した。ゲームが誕生した'70年代の様式は，輝ける未来や宇宙時代の雰囲気を感じさせる。それと同様に，昔のゲームのいわゆる「ドット絵」は，わくわくするようなイマジネーションに溢れていた。この部屋は，ある未来のような過去のような場所だが，そのどちらであろうと，プレイヤー(霧島あさひ)はきっとわくわくするようなゲームで遊んでいるに違いない！ ゲームの表現手法は時代とともに変われど，ゲームは常にそんな気分を提供してくれるものであってほしい。

表紙 由水 桂
ナムコ出身のCGデザイナー。

代表作
■ゲーム
『ワールドスタジアムEX』オープニングムービー
『RAGE RACER』オープニングムービー
『風のクロノア』ムービー
『R4-RIDGE RACER TYPE 4-』オープニングムービー
『ACE COMVAT3 electrosphere』コンセプトデザイン
他
■スチール(雑誌表紙／グラビア)
『NextGeneration』『週刊ファミ通』
『GaZO』『g M』『PlayStarion Magazine』
『EDGE(2000 namco calender)』
他
■アニメーション
『DRIVIN' WOMAN』『ピーナッツ』
『THE ESCALATOR』
他
■著書
『LightWave 3D スーパーテクニック』
ソフトバンク パブリッシング刊

http://home.att.ne.jp/red/yosimizu/

さて，この本が発売されるころにはPlayStation2が発売されているはずだが，第2次次世代ゲーム機戦争での大本命であることはいまさらいうまでもない。「戦争」というほどの争いすらなく，実際には2番手争いしか展開されないかもしれないくらいだ。他陣営ではようやくDreamcastでゲームが揃い始めたという段階，DolphinやXBOXはまだ霧の中。Dreamcast2ももちろん同様だ。

結果論を話してもしかたないのだが，第1次ではSCEの圧勝となった。要因はなんだろう？　参入ソフトハウスの数と熱心さだろう。タイミングよくキラーソフトが出たとかいうのもその一端にすぎない。セガは自社自身が非常に優良なゲーム制作メーカーであり，任天堂でもこれは同じだ。任天堂のクオリティでゲームを作れるメーカーはほぼ存在しない。SCEではGTをはじめ，いくつかのヒット作を擁しているものの，ゲームのクオリティに関してはさほどでもない。独占的な強さを持たないということがサードパーティの参入を促しているという，ちょっと変わった状況にあった。

GTについては，従来の「ゲーム」の枠を超えたところでシミュレータとして評価された初めての作品だ。PlayStationの本質がゲーム機というよりシミュレータの性格の強いものだったのだが，その象徴的作品だといっていい。従来型のゲームを作り慣れていないからこそ，陥りがちな轍を踏むこともなく，まさに斬新な作品として登場することができたのだろう。単なるゲームとして評価するにはゲーム部分はお粗末だし，結果としてはちょっと売れすぎという気がしなくもない。従来のゲームユーザー以外への訴求力を持っていたというのは事実だろうが，具体的にどこがどうウケたのかという分析はあまり意味がないだろう。ある意味「勢い」というしかない。

が，GT2ではシミュレータとしてのアルゴリズムなどは改良されたのだろうが，作品全体としてのシミュレータ性は極端に下がってきている。「敵車のアルゴリズムを作り込む時間がなかった」と関係者からの声も聞いているが，少なくとも前作と同じなら問題はなかったものを，変に色気を出してゲーム寄りの位置に近づけようとして失敗していることがわかる。実際にプレイした人なら誰でもわかるだろうが，触っていない人のために解説すると，最初のGTでは基本的にレース展開もシミュレーション路線というか，特に小細工はないのに対し，ゲーム調整が働いているのだ。つまり，トップに立つとエンジンのフケが悪くなり，

西暦2000年，次世代は始まっているのか？

最下位に落ちるとどんどんスピードが乗る……という展開だ。昔，ナムコが失敗した路線を忠実にたどっているようにも思われる。さじ加減次第ではゲーム性を上げることもできるのだろうが，結果としては，最終ラップまで全車団子状態でゴールに近づき，最後の最後でいかにして出し抜くか……という部分にゲーム性が集約されてしまう。理論上，それが当然の結果ではあるのだが。

実際のレースでそのようなことがありうるだろうか？　スリップストリームが利かないことが致命的となる競輪ではあるまいし（と，横内君は昔競輪のゲーム作りたがってたけどなあ），「リアルドライビングシミュレータ」と冠して発売されている作品がこれではマズイだろうと思うのは誰しも同じではないだろうか。ではシミュレータ指向とゲーム指向は相容れないものなのだろうか？　そうではない。ゲームにオートレベルコントロールがあるのがいけないというのではない。グラディウスではこちらの装備によって，敵の攻撃が激化していったが，それがゲーム性を下げる結果とはならなかった。格闘ゲームでは「ぜってーー卑怯！」と思われる攻防をCPUキャラがしでかしても，総力を挙げて叩きのめすほうに熱中した人も多いはずだ。が，それと同じようなことは起きなかった（少なくとも私の周辺では）。理不尽さが違うのだ。プレイヤーのパラメータを操作して弱くするような処理ほどプレイヤーの意欲を削ぐものはないだろう。要は，プレイヤー側の能力を制限してはいけないという単純なことが理解されていないだけの話だ。

少なくとも逆のかたちでの実装という手はあったはずなのだ。全車を抜き去ってトップに立てば，敵車全体がペースを上げて追ってくる……なんてのは実際のレースでは当然の展開だし，有力なライバルが序盤でスピンでもすれば，ペースダウンの指示を出さないピットはないだろう。ずいぶんと理不尽さも少なくなるし，オートレベルコントロールといった操作すらシミュレーションの結果として実装が可能なことがわかる。シミュレータとしての実装を放棄したことがすべての原因である。

ゲームに熟達すれば，ゲーム側の各種パラメータはバランスを取るために高騰する。RPGでみんなレベルMAXになってしまったら，よりMAX値を上げてさらに強い敵を出さざるをえない。こういったゲームにおけるインフレーションの原則を理解していないと，そしてこれを超えていかない限り次世代のゲームはありえない。そして，より高度な次元でのシミュレーションはひとつの鍵であるはずなのだ。





特別企画

PlayStation2の研究〈序章〉

次世代ゲーム機の本命，PlayStation2。発表と同時に多くの人を驚かせたハイスペックマシンだ。話題の機能は多いが，なんといっても注目はEmotion Synthesisという概念を引っ提げて登場した128ビットRISCプロセッサ「EmotionEngine」とEmbededRDRAMを2560ビット内部バスでつないで16個のレンダリングエンジンを並列動作させる「GraphicsSynthesizer」だろう。







今回は1年前の予備研究から多少更新された情報をもとにして，ひょっとしたらこれからPlayStation2のソフトを作るかもしれない人々のために，CPU部分を中心としたシステム構成を確認していこう。

Emotion Engineとはなにか

西川善司 Nishikawa Zenji　中野修一 Nakano Shuichi

ついに発売された（はず……）PlayStation2。そのスペックや性能はいまだに「なんかすごい」ということだけで一般のパソコン好きたちにはどうも具体像が見えてこない。現在公開されている情報を少し整理して，PS2の心臓部であり頭脳であるEmotionEngineのポテンシャルを探ってみることにしたい。すでに1年ほど前の1999春号で当時入手可能な資料をもとに，PlayStation2の予備研究を行ったわけだが，当時不明だった部分，疑問点として挙げていたもののうち，いくつかは明らかになった。まず，そのあたりを整理しておこう。

CPUとなるEmotionEngineは正真正銘の128ビットプロセッサだ。通常の整数演算などは64ビットみたいだが，レジスタ長を生かしたマルチメディア系の命令が追加されている（MMXのようなもの）。

画面解像度については，VGA解像度（640×480ドット）のほか，

- 1024×768ドット24ビットカラー
- 1280×720ドット24ビットカラー
- 1280×1024ドット16ビットカラー
- 1920×1024ドット24ビットカラー（インタレース）

などが可能とされている。

とはいうものの，やはり現実的にはインタレースVGAが標準となるものと思われる。テクスチャを随時メインメモリから取ってくるテクスチャスワップシステムはミドルウェアで実現されるという。パフォーマンスは落ちると思われるが，これでテクスチャ容量の問題は一応解決できる。しかし，ミドルウェア関係はすべて個別に有料（かなり高価と聞く）なので，実際にそういう処理を行うゲームが一般的になるとはちょっと思えない。

フレームバッファ（含バックバッファ）に2.4MB，テクスチャバッファに1.6MBという構成が一般的と思っていいだろう。シーンごとのテクスチャマネージメントをしっかりやればさほど問題になることもあるまい。ピクセルフォーマットをはじめ，かなり柔軟な構成のようなので，バックバッファなしのデザインも不可能ではないと思われるが，レンダリングが厳しそうなので最初から無理をする人はあまりいないのではないだろうか。

なお，あの時点でのPS2が250MHz動作という記述があったが，250MHzだったのはデモ機の1台だけで，ほかは300MHzで動作していたということだ。

PS2の全体像

図1はPS2のシステムブロックダイアグラムだ。それぞれの線はバスを表し，そこに添えられた数値はバス幅（ビット数）を示している。また，RDRAMやSDRAMのところに記載された容量値は最大容量を表している。実際にそれだけのRAMが積まれているわけではないので注意。

EmotionEngineのメインボーンとなるバスは128ビットだがすべての周辺I/Oと128ビット接続されているわけではなく，I/Oプロセッサとは32ビット，GraphicsSynthesizer（GS）とは64ビット，RDRAMとは16ビット×2CHという配分で接続されている。

I/Oプロセッサの部分はPS2ネイティブで動作させているときはI/Oプロセッサとして活動するが，PS1互換モードで動作しているときにはこのI/OプロセッサがCPUとして活動する。その場合のメインメモリとしてはI/Oプロセッサに32ビット幅で接続されたSDRAM部分が使われる。

グラフィック表示はPS1互換モードでもGSを通して行う。すでに，各方面で伝えられて周知の事実となっているが，PS2本体側の設定を行うとPS1のソフトでもテクスチャにフィルタリング処理がされるのはこの構造のおかげでもある。

PCMCIA interfaceはPCカードスロットのインタフェイスだが，これは意外にもCardBusのような32ビットバスではなく，ごくレガシーな16ビットタイプが使われている。SCEではPS2のPCカードスロットは「通信ポート」と説明していることからモデムやネットワーク関係の機器以外の接続は想定していないものと思われるが，TYPE Ⅲスロットを採用しており，ネットワークへの展開時にはHDDは不可欠とする見解もうかがえるため，HDDのようなストレージ系カードとの接続も可能となるのだろう。ネットワークのバックボーンができてくれば，TYPE Ⅲのネットワーク＆HDDコンボカードのようなものが発売されるのではないかと予測される（あるいは，HDD内蔵の外付けセットトップボックスとの接続インタフェイスかもしれない）。

ROMにはBIOS，IPLやDVDプレイヤーなどの基本ソフトウェアが記録されており，必要に応じてこれらを起動して使うことになる。

サウンドプロセッサ（SPU2）は基本的にはPS1のものを単純に性能強化したものが使われており，サウンドチップ自体の基本機能は同じ。音源方式もADPCM。ただし同時発音数は2倍の48声に高められている。ソフトウェア側でチャンネルミキシングダウン処理を行えばそれ以上の同時発音数や，あるいはMIDI音源のようなウェーブテーブル音源も実現可能となっている。また，知らない人はいないと思うがPS2はDVDプレイヤーにもなるため，そのデジタルオーディオ出力端子として光端子が1基実装されており，ココへの出力もこのSPU2が担当している。

EmotionEngineの構成

それではEmotionEngineの内部について見ていくことにしよう。

図2はEmotionEngine内部のブロックダイアグラムだ。

EmotionEngineは3つのプロセッサコアからなるCPUだ。ひとつはいわゆるコプロセッサつきのMIPS系のRISCプロセッサ，あとの2つは2つのVPU（Vector Processor Unit）コアだ。こ

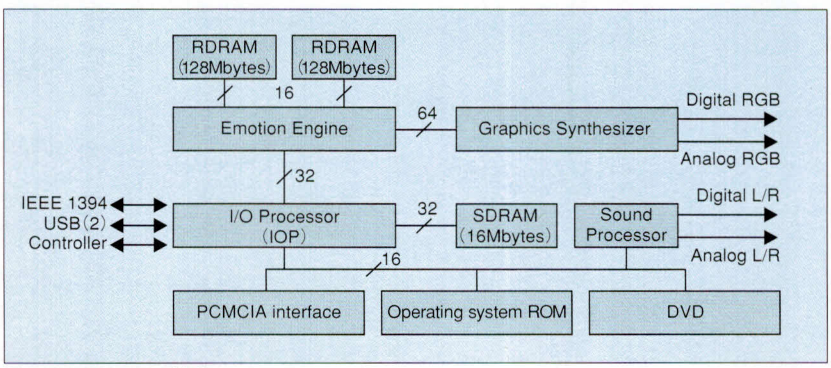


図1　PlayStation2の構成

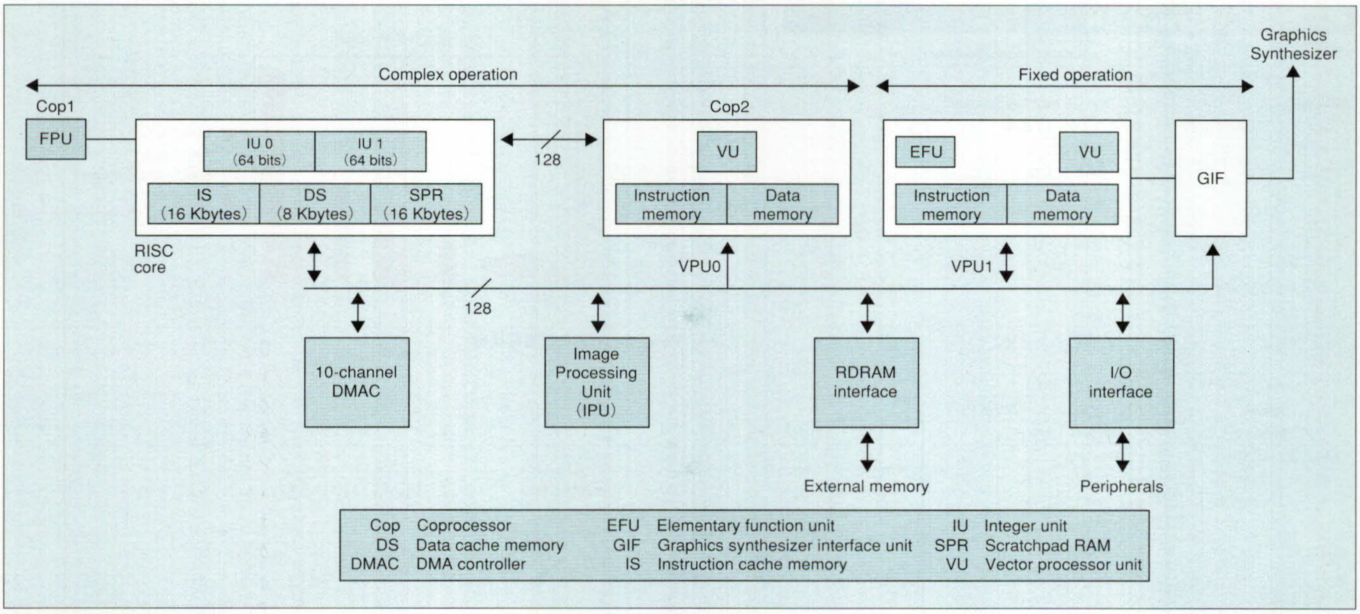


図2　Emotion Engineの構造

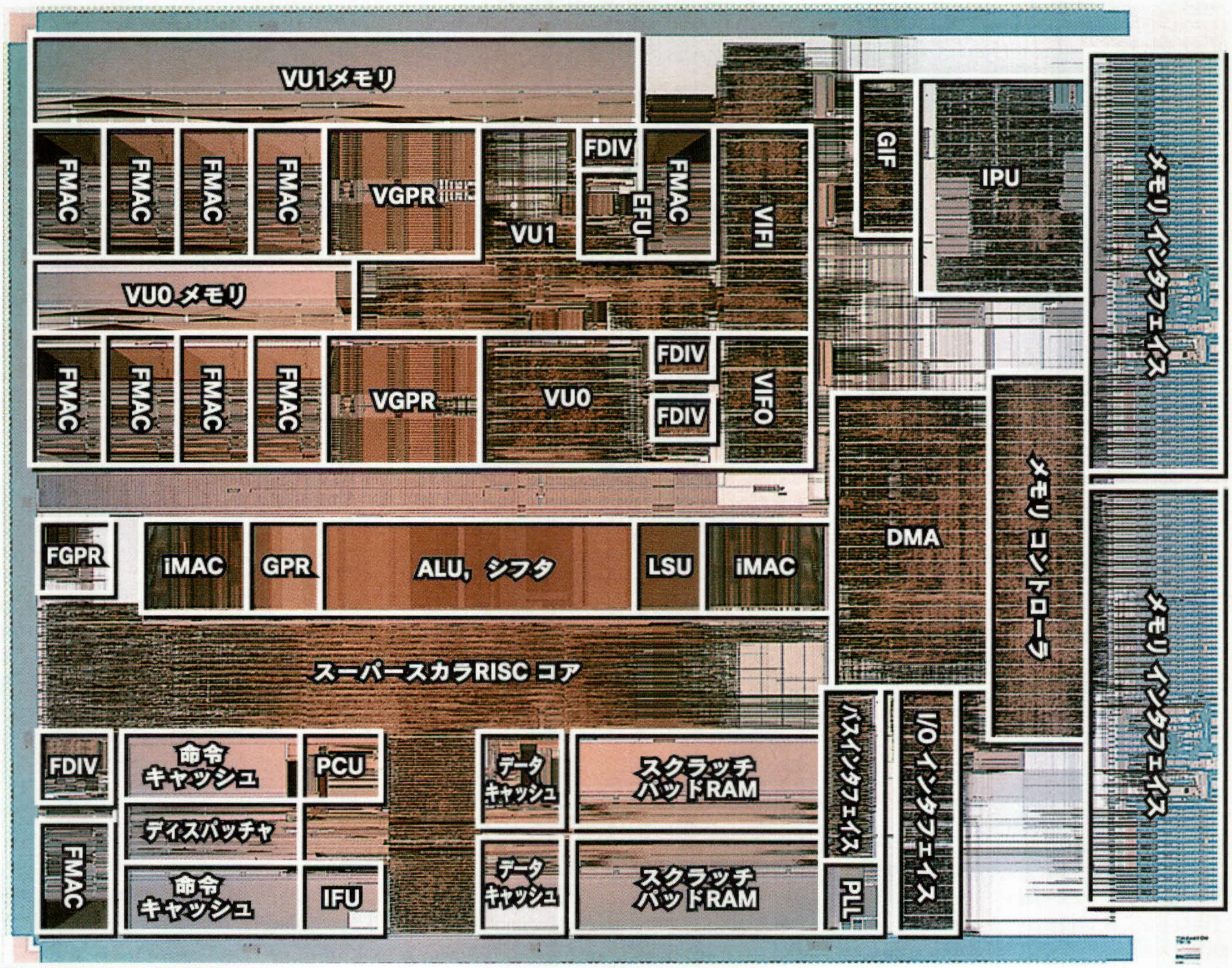


図3　EmotionEngineの内部ブロック

表1 EmotionEngineの仕様

CPUコア	128ビット RISC(MIPS IV-subset)
クロック周波数	300MHz
整数演算ユニット	64ビット(2-way superscalar)
マルチメディア拡張命令	128ビット×107種類
GPR(整数レジスタ)	128ビット×32本
TLB	48ダブルエントリー
命令キャッシュ	16KB(2-way)
データキャッシュ	8KB(2-way)
スクラッチパッドRAM	16KB(dual-port)
メインメモリ	32MB(RDRAM×2ch@400MHz)
メモリバスバンド幅	3.2GB/秒
DMA	10-channel
コプロセッサ1	FPU(FMAC×1，FDIV×1)
コプロセッサ2	VU0(FMAC×4，FDIV×1)
マイクロ命令用メモリ	I：4KB/D：4KB
ベクトル演算器	VU1(FMAC×5，FDIV×2)
マイクロ命令用メモリ	I：16KB/D：16KB
浮動小数点演算性能	6.2GFLOPS
座標変換＋透視変換	6600万ポリゴン/秒
＋光源計算	3800万ポリゴン/秒
＋フォグ	3600万ポリゴン/秒
曲面生成(ベジエ)	1600万ポリゴン/秒
IPU	MPEG2 マクロブロックレイヤデコーダ
画像処理速度	15000万 ピクセル/秒
ゲート長	0.18ミクロン
コア電圧	1.8V
消費電力	15W
メタル配線層数	4
総トランジスタ数	13.5M Tr.
ダイサイズ	$240mm^2$
パッケージ	540ピンPBGA

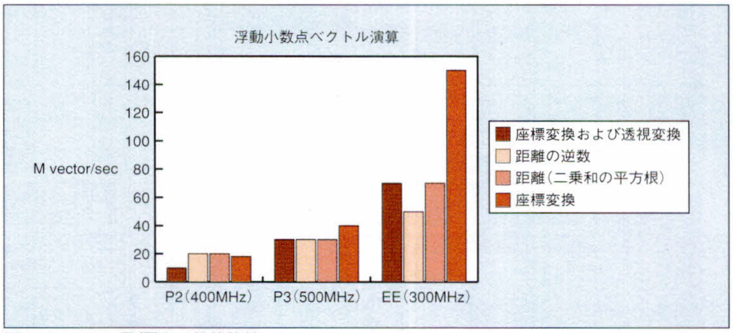


図4 Pentium II/IIIとの性能比較

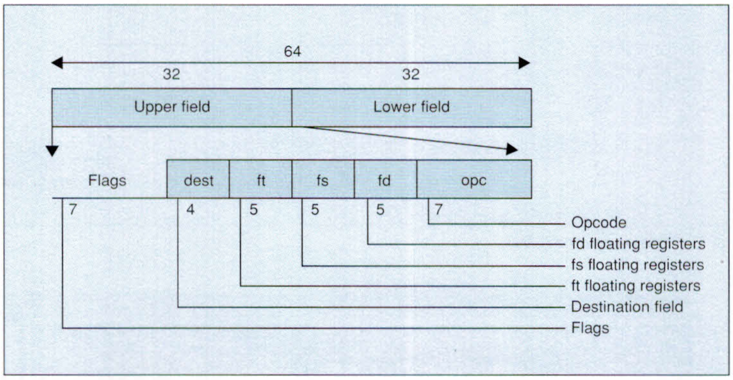


図5 VPU用の命令フォーマット

のほか，IPUと呼ばれるMPEG2アクセラレーションコア，10CH独立駆動型DMAなどのコア，RDRAMインタフェイス，I/Oインタフェイスなどもが EmotionEngineに含まれている。パソコンでいう「チップセット」的な役割もEmotion Engineが果たしていることになるわけだ。

もう少し詳しく見てみよう。図3はEmotion Engineの内部構成図（予想図）だ。1昨年発表されてから少し（約300万トランジスタ）トランジスタ数が増えたようで，実際のレイアウトとは多少違う可能性もある。

全体の略語の意味は以下のとおり。

- VU0　ベクトルユニット0
- VU1　ベクトルユニット1
- VIF　ベクトルユニットインタフェイス
- FAMC　浮動小数点積和演算器
- FDIV　浮動小数点除算器
- GIF　GraphicSynthesizerインタフェイス
- IPU　画像処理ユニット
- VGPR　汎用ベクトルレジスタ
- EFU　基本関数ユニット
- FGPR　汎用浮動小数点レジスタ
- IMAC　整数積和演算器
- ALU　演算器
- LSU　ロードストアユニット
- PCU　プログラム制御ユニット
- IFU　命令フェッチユニット
- DMA　ダイレクトメモリアクセスコントローラ
- PLL　クロック分周器

命令キャッシュは16KB，データキャッシュは8KB，スクラッチパッドRAMは16KB搭載されている。ざっと見てそれぞれ2つのブロックに分かれているのは2ウェイセットアソシアティブのウェイ単位かと思っていたのだが，実際には奇数ビットと偶数ビットで分けられているためだ。この図では奇数ビット/偶数ビットで上下分割されているが，さらにそれぞれは内部で左右に2分割され，センスアンプやロードアライナなどは2つずつ設置されている。

VPU部分のRAMは一塊に見えるが，これは命令/データで分離されている。VP0ではそれぞれ4KB，VP1ではそれぞれ16KBだ。VPUはそれぞれのローカル命令メモリに読み込んだ命令を実行するようにできている。ベクトルユニットではそれぞれ内部にベクトルレジスタ（実数：128ビット×32，整数：16ビット×16），浮動小数点積和器×4，浮動小数点除算器，乱数発生機，ロードストアユニットなどを備えている。VP0とVP1ではコプロセッサインタフェイスの有無，EFUの有無，メモリ容量などの差がある。基本的には同じものだが，位置づけはまったく異なっている。

また，一見，VP0側にFDIVが2個ついているように見えるのだが，実際にはVP0，VP1とも4個の積和器と1個の除算器を備え，1個の積和器，除算器などのセットになったEFUがVP1側についているという構造になっている。

VPUのひみつ

PS2のEmotionEngineがやたら「すごい」といわれているのはどの部分かといえば，やはりVPUの部分だ表1。「同クロックのPentium IIIの10倍の演算性能」（図4）なんていう形容がひとり歩きしてみんな目をぱちくりさせているのだが，確かに，EmotionEngineにとって都合のいい比較の仕方をすれば実際にそうならないこともない。もっと一般化すれば，Pentium IIIの倍といったところだろう。同じグラフ上にPowerPC G4がないのが残念だ（アップルの主張では500MHz時にPentiumIII/800MHzの平均2.2倍の浮動小数点演算性能）。

EmotionEngineの実数演算はMIPSコアのFPUで行うこともできるが，3D系処理で頻繁に発生するベクトルの積和計算はVPUコアで行えば，1クロックごとに結果を出すことができる。

たとえば，

x3=x0*x1+x2
y3=y0*y1+y2
z3=z0*z1+z2
w3=w0*w1+w2

という計算も全部まとめてスループット1サイクルでできてしまうということだ。こんな芸当ができるのもVPUはSIMD-VLIWアーキテクチャ(*1)を採用しているからにほかならない。

VPUでは128ビットの実数レジスタが32個利用可能になっているが，その1個のレジスタにつき，32ビット長の浮動小数をx，y，z，wの4つのフィールドに分けてストアさせて演算を行わせることになる。実際の演算もレジスタをx，y，z，wフィールドに分けて，各フィールド専用の演算ユニットで並列で処理される。これが1クロックで実行される秘密だ。

なお，Pentium IIIでも同様な機構は持っているが次の結果を得るまでに2クロックかかってしまう。最近では，EmotionEngineの演算性能について，整数演算ではPentium IIIの半分，浮動

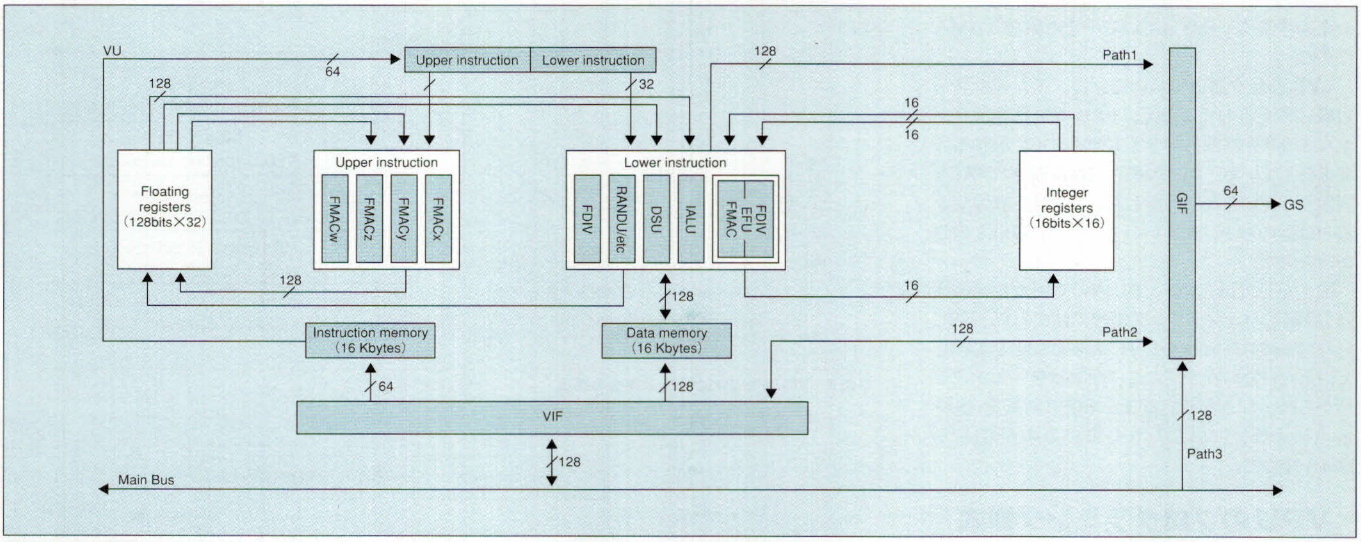


図6　VPUの構造

小数点演算性能では2倍といったいい方もされているようだ。

さらに，VPUでは，

$$(x, y, z, w) = (x0, y0, z0, w0)\begin{pmatrix} m0 & m1 & m2 & m3 \\ m4 & m5 & m6 & m7 \\ m8 & m9 & ma & mb \\ mc & md & me & mf \end{pmatrix}$$

といった投影変換やアフィン変換で使われる4×4行列とベクトルの乗算を8クロック，4×4行列同士の乗算を16クロックのスループットで実行できる。

（＊1）SIMD=Single Instruction/Multiple Data。1命令で複数のデータを処理する命令系。VLIW=Very Long Instruction Word。1語に複数の命令語を含ませる命令系。

VPU命令語の様式

VPUの命令語は図5のような64ビットフォーマットになっている。fs～fdは演算元になるオペランドレジスタ番号，destは演算結果の格納先に対応する。内部は32ビット命令で構成されており，これが2つセットになって64ビットVLIW命令としてVPUに渡されることになる。

命令がUpper fieldとLower fieldと2つの命令語に分かれているのも特徴で，それぞれ同じクロック内で並列に実行サイクルにかけられる。Upperは積和算器（FMAC）用の命令語，Lowerはロード，ストアといったメモリ操作系，さらに除算器（FDIV）を駆動するような命令語がくる。FDIVを駆動した場合に限り，実行クロックは7～13クロックになるが，それ以外は1クロックで実行が完了する（EFU命令も除く）。

演算結果はレジスタに退避する以外に，GSへ転送したり，VPU内のローカルメモリに退避したりすることができる（図6）。

VPUを効率よく動かすための工夫

VPUはメインメモリのアクセスサイクルとは完全に独立して動作しているという点も，VPU

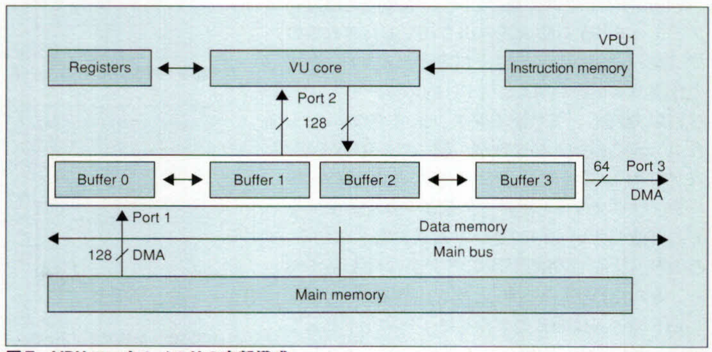


図7　VPUローカルメモリの内部構成

EmotionEngineの使用例

EmotionEngineの活用例として挙げられている水面の反射屈折を行ったデモの概要を見ていきたい。

この処理は6つのステージに分かれる。

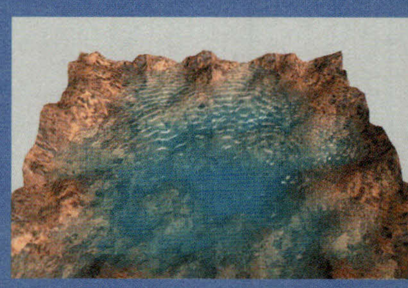
1) 地形の生成
2) 波紋の生成
3) 地形のレンダリング
4) 水面のレンダリング
5) 屈折のレンダリング
6) 水面の反射などのレンダリング

これらのうち，1)，2)に関してはかなりデータ量が多く複雑な演算が必要なので，あまりVPU向きではない。データはメインメモリに置き，演算はCPUで行うことが望ましい。CPUによって3Dディスプレイリストが生成されれば以降はVPUが処理を行うことになる。

地形データなどはVP1のデータRAMに格納される。VP1のデータRAMは16KBの容量を持っている。1頂点のデータ量が16バイトだとすると，1024頂点が格納できる。これはSTRIPで処理されるとすると32×32頂点のマトリクスとなることがわかる。サンプル画像を見ると，なるほど32×32頂点っぽい感じだ。

4)から6)については，テクスチャのモジュレーションや光源計算などで複雑なベクトル演算が必要になる。こういった用途にはVP0をCPUコアのコプロセッサとして活用することが望ましい。

かくして，それぞれの演算器が有効に使われて複雑な地形を考慮したリアルタイムの水面反射屈折シミュレーションが完成することになる。なお，マトリクスが32×32より大きければ地形を分割して必要なだけ処理を繰り返せばよい。50万頂点くらいは扱える性能を持っているので，1000頂点単位程度の処理はなんでもないだろう。



の高速性能に一役買っている。その秘密はDMAにある。

VPUの命令語は実行のたびにメインメモリから取ってくるのではなく，VPUコア自身はローカルメモリから独自のタイムサイクルで取り出して実行している。周りの都合には左右されず好き放題やっているという感じだ。この好き放題できるのは誰のおかげかというと，そう，DMAのおかげなのだ。

図7を見てほしい。VPU内のローカルメモリは4段階にバッファリング処理されており，VPU自身はこのローカルメモリにある命令語を実行しているだけなのだが，ここに命令語をメインメモリから持ってきたり，ここから演算結果をGSやメインメモリに戻したりしているのがGIFやDMAなのだ。

VPUのプログラミング様式

VPUの命令語は普通のCPUの機械語とは違いメインメモリから順次読み出されて実行されるのではない。メインメモリ上に書かれたVPU専用の仕事リストをDMAがVPU内のローカルメモリに取り出し，VPUインタフェイスユニット(VIF)がこのヘッダを理解，続いてVIFがVPUコアに対してその仕事リストに書かれた仕事(VPUプログラム)とデータをローカルメモリを介して渡していく……という様式になっている。このリストには要所要所にタグがつけられており，それを「VIFコード」と呼んでいる。

VIFコードは図8のようなフォーマットとして定められており，128ビット単位のディスプレイリストの最初の32ビット部分に埋め込まれた感じになっており，その他のデータ構造はいわゆるチャンク式のリスト構造になっている。

一例を示すと，VIFコードでまずVUPをどう動かすかというプログラム自体を読み込み，続くパケットで処理すべき頂点リスト，テクスチャなど，必要となるデータをパッケージしてVPUに渡し，すべて受け取り終わると処理を実行する。

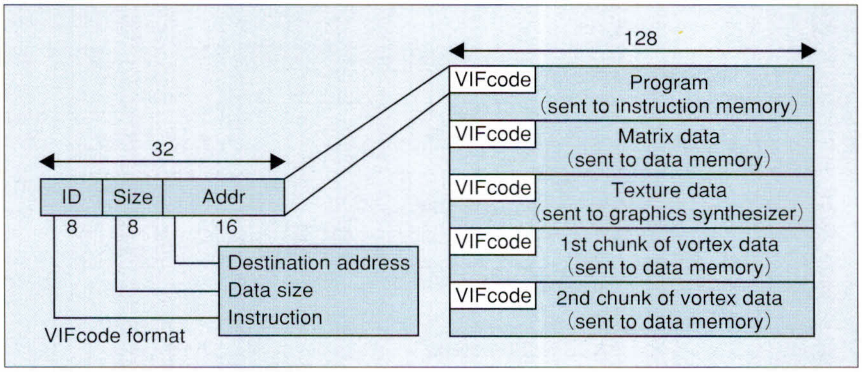


図8　VPUデータパケットと命令の構造

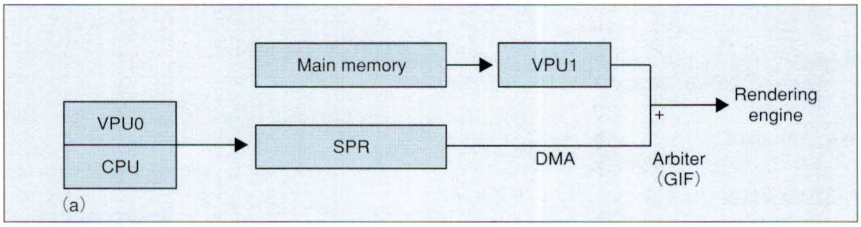


図9　パラレルVPUモード

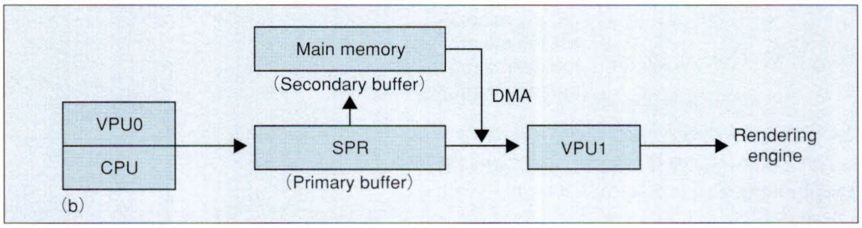


図10　シリアルVPUモード

このVPUの仕事リストのうちでもっとも多用されるであろうものが3Dディスプレイリストというものだ。その名のとおり，3D座標の並んだ頂点リストなどを透視変換し，GraphicsSysthsizerが処理できる2Dディスプレイリストに変換して渡すことになる。

基本的にVPUは大量のデータに次々と同じ処理を施していく。処理すべきデータがなくなったりすると，GIFなどが自動的に次のデータを補充するようになっており，VPU自体は処理に専念できるような構成が取られている。

VPU，2つの動作モード

EmotionEngineのVPUはVPU0とVPU1の2つがインプリメントされている。基本機能は同

PCとの比較

PCとPlayStation2ではどの程度性能が違うのだろうか？　アーキテクチャが違うので一概には比較できないのだが，PCで作成したプログラムとPlayStation2で実行可能なプログラムの差が気になる人もいるだろう。PCでどの程度のことが可能なのか個々のパーツだけ見れば一長一短という感じだが，PlayStation2全体の設計の特徴として，きわめて動作の並列度が高いのだ。

CPU性能は整数演算能力でPentium IIIがEmotionEngineの倍，浮動小数点演算能力だとEmotionEngineがPentium IIIの倍といったところだろう。グラフィック性能は単純描画性能では，

GraphicsSynthesizer　2.4GPixel/Sec
GeForce256　480MPixel/Sec
RIVA TNT2 Pro　286MPixel/Sec

といったフィルレートの差で，GraphicSynthesizerの強力さがわかる。が，テクスチャを貼っていくと，GraphicsSynthesizerでは値が半減する。フィルタリングなどを加えていくとそれに比例して性能低下が見られる。PlayStation2では基本解像度が低いため，フルスクリーンアンチエイリアスは標準となる機能だろう。一定の画質にするにはかなりの処理が必要になる。

ポリゴン性能を見てみよう。

GraphicsSynthesizer　75MPoly/Sec
GeForce256　15MPoly/Sec
RIVA TNT2 Pro　6MPoly/Sec

GeForce256での実測値では5MPoly/Sec程度(3D Mark2000使用時)となる。PC系のビデオカードではテクスチャを貼る枚数を増やしたり，複雑な処理をしても1パスでサポートされていたりなど，高度な表現を行った場合でも性能低下が少ない。

また，ハードウェアT&L対応ビデオカードを使用すると，VP1に相当する部分をビデオカード側が担当する感じになる。この場合，総合的な演算能力はほぼ同じくらいになるものと考えてもよいかもしれない。

じものと考えていいが，前述のとおりVPU1のほうにはFMACとFDIVひとつずつからなるEFU (Elementary Function Unit) と呼ばれる演算ユニットが付加されていて，VPU0よりもやや高度なことができるようになっている。まぁ，いずれにせよ基本的な演算能力は同等だ。

さて，PS2では，この2つのVPUをどのように動かしているのだろうか。

通常の使用状態では，この2つはまったく違った用途で使われる。VP0はCPU側から見ると2個目の浮動小数点演算コプロセッサのようになっており，物理演算など複雑で大量の浮動小数点演算を必要とするプログラムの実行中にCPUコアから呼び出されることになる。

それに対して，VP1はもっぱら3D座標変換処理をこなすように作られている。

とはいえ，動作はあくまでプログラム次第でどうにでもなる。それぞれ独立で動くことを特徴として作られているのだが，両者を協調的に動作させることもできる。これには2つの利用形態が用意されており，ユーザーは開発しているアプリケーションに適した任意の動作モードが選択できるようになっている。

その2つのモードを図解したものが図9，10だ。

(a)は「パラレルモード」でVPU0とVPU1にまったく別の描画リストを処理させることができるため，レンダリングエンジン側から見れば，非常にコンスタントに結果がもらえる方式だ。この方式をサポートするためにGIFではデータパスを二重化してある。両方のVPUで別々に計算したデータをマージして表示するようなことが可能だ。

VPU0は処理した結果をスクラッチパッドRAM (SPR) に格納しておき，レンダリングエンジンはVPU1からデータがこなくなったらSPRのほうから結果を取り出す……というような手順で処理が進む。

(b)は「シリアルモード」で，VPU0をVPU1のプリプロセッサとして働かせるモードだ。この場合，VPU0の演算結果はSPRに格納され，VPU1が暇であればそのままその結果を受け取ってさらに演算を進め，VPU1が忙しくてすぐに処理できない場合はVPU0の演算結果はSPRにしばらく保存される。SPRが満杯になってしまった場合は今度はメインメモリ側の二次バッファに結果を退避させる。

どちらの方式もユーザーのアプリケーションの構造に適合した場合は複雑な3D CG処理を効率よくできることになる。

以上がPS2の左脳ともいえるEmotionEngineのVPUの概要だ。

各ハードウェアの独立性を高くし，汎用性を第一に設計された高速逐次処理を基本としたPCのシステムとはだいぶ異なっているために，スペックからはなかなか本当の潜在能力が見えてこないが，いずれ，その本当の姿はこれから登場してくるだろう，たくさんのアプリケーションが明らかにしてくれるだろう。

参考文献：DESIGNING AND PROGAMMING THE EMOTION ENGINE, IEEE MICRO1999

他陣営の動向

さて，第2次次世代機抗争はなにもPlayStation2の勝利に決まったわけではない。非常に優位ではあるものの，世の中なにがあるかわからない。

対抗馬の筆頭である任天堂の次世代機Dolphinの動向はいまひとつはっきりしない。CPUに使われるはずのPowerPC（コードネームGekko）は当初アナウンスされていた仕様に変更が加わるのではないかと思われる。256KB程度のL2キャッシュを載せ，600MHz程度で駆動するものが搭載されると噂されている。

グラフィック部分を担当していたArtXの詳細についてはさらに不明点が多い。PC用のチップセット，Aladdin7でArtXのテクノロジーが採用されることが公開され，その一端くらいはうかがえるようになった。Aladdin7の特徴はなんといっても，メインメモリのバス幅を128ビットに拡大していることだ。従来型のSDRAMを使ったままメモリバンド幅を2倍に拡大している。Intelのi820がDirectRDRAMを使って1.6GB/Secのメモリ帯域を実現したのに対し，SDRAM DIMMを2本ずつ使うという明解な路線で同程度のメモリ帯域（クロック次第ではさらに広い）を達成している。このため，従来型のメモリ共用アーキテクチャ（SGI Cobaltは置いといて）では性能面で妥協せざるをえなかった部分を妥協なしで高性能を狙える構成にできる（とはいえ，一般的なビデオカードには及ばないのだが）。

さらにArtXがATIテクノロジーズに買収され，Slot1用チップセット（ノースブリッジ）にいわゆるハードウェアT&L機能を載せたものが登場するという。おそらく同社のビデオチップをオンボード搭載して，低価格高性能マシンが実現できるような構成となるのだろう。

このようにPC関連メーカーには大人気のArtXなのだが，描画部分にしてもT&L部分にしても，どの程度のパフォーマンスなのかというのはいまひとつはっきりしない。製品の位置づけとしては当然，PlayStation2を上回る性能でなくてはいけない。任天堂はSCE/東芝のEnbededRDRAMのさらに上を行くEnbededSRAMテクノロジーを採用する。1トランジスタでSRAMを構成するMOSYSの技術は非常に高密度で超高速なメモリを低価格で実現する可能性が高い。現在のチップ資料では従来の6トランジスタタイプSRAMと同速度で動作し，400MHzにまで対応，256ビットバス幅までサポートと，とてつもない可能性を感じさせるスペックが羅列されている。

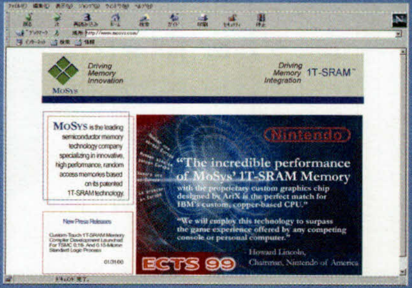
対抗の勢力として無視できないのがマイクロソフトの動きだ。コードネームX-Boxとして知られるセットトップボックスが今年中にも出てくるのではないかとまことしやかに噂されている。早ければすぐにも発表されるかもしれない。

噂段階のものなのでかなり確度は怪しいが，X-BoxはDVD-ROM搭載のセットトップボックスでSpitfireコアのAthlonプロセッサ，NV15相当のGeForce256を中心として構成されるという。OSはWindowsCEともWindows98とも，新規のものとも噂されるが，DirectXを動作させるものであることははっきりしている（Win32アプリが動くWindowsCE？）。

メモリ量などはまったく不明。HDDを搭載すると見られるので，他社製品とは違った位置づけができるかもしれない。問題は価格だが，これは推測しようがない。現状のWindowsCEマシン（Dreamcastは置いといて）くらい高いものであったら売れないだろうが，搭載するといわれるハードの価格からすると低価格は難しそうだ。PlayStation2はあの仕様でちゃんと利益が出る価格設定だというから，数量次第では相当価格を抑えられるのかもしれないが，現状では推測不可能だ。

X-Boxは標準仕様というかたちで出てくると思われるので，ハード仕様自体は今後アップグレードなどもありうるだろう。各社それぞれに個性のある製品が出てくるかもしれない。パフォーマンスは現在のPCゲームクラスのものが動けば，とりあえず問題はないだろう。

これを受けて噂されるのはセガのDreamcast2だ。X-Box仕様で，SH系CPUを積み，Dreamcastの資産を継承するという。AthlonとSH5のデュアルプロセッサではコストがかかりすぎる気はするが……。グラフィックチップはGeForce256ではなく，PowerVR3（？）が搭載されるという噂もある。眉唾な話といえば，この上ないくらいだが，マイクロソフトがからんでいるとなると，どう転んでいくか見当がつかない。共通規格であるゆえにアクセラレータの対応が進むことは明白であり，強力なアクセラレータも随時投入されるだろう。開発環境なども現状のPCと共通にでき（Dreamcastのときもそのはずだったんだが），開発者にはうれしい製品となるかもしれない。ガセでないことを祈りつつ，次なる情報を待つとしよう。



Your Memories, My Memories

田中順子　*Tanaka Yoriko*

おいしい料理の作り方



家庭料理って本当にいいものですよね。お母さんの作る料理は，素朴で地味なものばかりです。しかし，なんともいえない美味しさがあります。特に学校などに持っていくお弁当は自慢でした。そこで，私も家庭料理というのを作ってみたくなり，少し母から教わることにしました。






しかし，料理を美味しく作るのって難しいですね。教わったとおり作っても美味しくなかったなんてことがよくあります。いったいどこが悪いのかな？　なんて考えてみても原因を発見できないまま，また同じ失敗を繰り返してしまいます。しかし慣れてきて包丁の使い方や火加減などがわかってくると，不思議なことに料理の味ってよくなるようです。料理の美味しさの秘密って，こういった地味な作業の積み重ねなのかもしれませんね。



完成した料理を眺めていると苦労した分，満足感も一入です。少し形が崩れていたり，味が少しくらい味覚とあわなくても，完成した料理の前では些細なことです。そして料理には，真心という隠し味こそがもっとも重要なのです。



感じるリアルさとゲームの面白さ

古村聡　Komura Satoshi



ここしばらくのシミュレーション系ヒットゲームといえば，ダンスダンスレボリューションに次いで，この「電車でGo!」でしょう。こいつは京浜東北線快速や新幹線の運転が可能になった「電車でGo! 2」

コンピュータゲームのひとつの究極のかたちである「シミュレータ」。手軽に味わえる「本物感覚」が決め手です。では，その本物感覚はどこからやってくるのか？　ここではプレイヤーが感じるリアルさについて考えてみましょう。

ああ，いっそのこと「これ以上はなにがあってももう面白いものは絶対に出ません！　論理的に不可能で」ってゲームが現れてくれたらいいと思いません？　そうしたら，一生，そのゲームだけを抱えて，それだけを見て生きて（もしかしたらそのまま死んで？）いけるでしょうに……。でも，現実は悲しいかな，やっぱり次から次にゲームは生産されて，販売されて，そしてまたそれを買ってしまうという……。

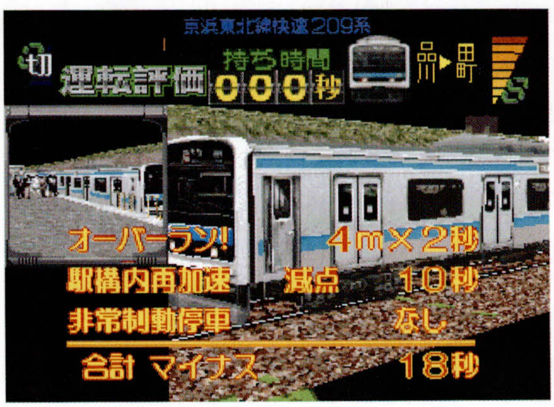
ここにマイクロソフトの「フライトシミュレータ」というソフトがあります。簡単にいうと，パソコンで，飛行機を操縦して空を飛んだ気分になるためのソフトです。これ，うちにいくつあると思います？　バージョン6.0，95英語，95日本語，98英語，2000日本語……5本ですよ，5本。で，好きなもんですから，この手のソフトは出たらまた買ってしまうんですよ。

最近私がプレイしているのは先ほどの，「マイクロソフトフライトシミュレータ2000」に，「Flight Unlimited3」，コンビニで安く売っていた「ぼくは航空管制官」，それにPSの「JETでGo！」……飛行機のシミュレーションものですね。あ，「電車でGo！2」もあったか。こっちは電車のシミュレーション（？）だけど。

なんかわざとらしくその手のものを集めてしまいましたが（これにはひこうきとかでんしゃが出てるとそのあいだうちのちびがおとなしくなるから，という海よりもふかーいわけがあるのです。おかげで18禁ゲームをプレイする時間があんまりなくてねぇ。とほほ。ま，少しはやってますが……），まぁ，私がもともとこの手のゲームが好きなのだ，という事情もあったりします。というか，個人的には，ある意味，この手のゲームこそがコンピュータゲームの究極であると信じている私なのです。理由は簡単で，パソコンを買ったら絶対作ってみたかったのがこの「電車のシミュレーション」だったからなのです（それもFM-7とかX1の時代です）。コックピットビューで自分でダイヤどおりに電車を動かすソフトを絶対作ってやろうと思った私なのでした。パソコン買って，すぐに「自分のプログラムでは風景どころか，四角形ひとつスムーズに動かせない」ことを理解して挫折しちゃいましたが。

ともあれ，私ならずとも，この「現実の現象をあるがままにコンピュータで再現する」そして，それを「自分では現実には体験しえないものを体験する」というのは，ゲームプログラムのためにコンピュータを買った人なら一度は夢見るものなんではないかと私は固く信じております。コックピットに座るタイプのゲーム。電車だったり，あるいはザクだったり，グフだったり，飛行機だったり。ええ，私，かつて，ガンダムのゲームが出る，と聞いて喜んでいたら，それがストIIタイプの「格闘ゲーム」と聞いて，即刻「キルユー！」と叫んだ熱い心の元少年です。ガンダムっつったら，コックピットでしょぉ！？　「させるかぁ！」っつってシート横のスティックをぐーっと，前に押さなきゃいかんでしょぉ！？　人として，男の子として！　負けるな，ザブングルは男のコぅ！　はぁはぁ。

さて，その意味で，アーケードで登場した「電車でGo！」というゲームはエポックメーキングというか，ツボ，でしたな。電車を運転できる，という夢実現系がついに実現したわけですから。そして，実際にプレイして，いままで体験したことのない感触にしびれましたねぇ。そう，皆さんもそうだと思うんですが，「電車は急には動けない，止まれない」という電車の重量感なんですよ。「電車でGo！」というゲームでは運転席にある，モーターを動かす制御装置であるノッチを引くと電車が動き出すのに，少しタイムラグがあるのです。そして「ぐぅーっ」とゆっくり動き出すのです。そして止まるときも，ブレーキを引くとゆっくりゆっくりと速度が落ちていく。決して急には止まらないのです。ちょっと勾配があったりすると，止まれ止まれとレバーを押せども，ずるずるずるずると走っていく。2000番台の京浜東北線，秋葉原駅なんて，凄く電車が重く感じませんか？　あそこは駅で止まろうと思っても，ちょっと自分の重みで進んでしまうのがしっかりわかるんですよ。こ，これが209系11両編成，人間を1000人乗せる重みなのか〜！　と実感できる。本当に新しい感触に感動したのであります。



電車というものの重量感をいやでも味わうことになるこのゲーム。ブレーキタイミングは本当にリアル？

これってホントにリアル！？

しかしですね，発売からずいぶん経ったせいもありますが，今日はふと考えてしまったのですよ。「『電車でGo！』ってホントにリアルなんだろうか？」

たとえば，CMじゃないですが，「電車でGo！」を始めると，つい，本物の電車に乗るときに，電車の先頭車両に乗って，運転手さんの運転を見てしまうのですよね（私はやりました）。見てると，確かにノッチを引いてから時間差があって電車は動き出す。ブレーキを引いても，ちょっとやそっとじゃ止まらない，重量感。そのままみたいなんですよね。たとえばJR南武線6両編成の209系は停止する位置150メートルくらいからブレーキを3目盛りくらいかけて徐々に弱くしていくみたいです（研究したぞ）。ただ，ブレーキを引いてから本当にゲームと同じくらいの感覚で止まるのか，という加減までは……わからないんですよね。なんでって，だって，どんなに「電車でGo！」をやり込もうが，交通博物館まで行って山手線205系シミュレータをやろうが（子供連れて順番を並ぶのは大変でしたよ。運転が終わってから子供を運転席から引きはがすのはもっと大変でしたが……），自分で「本物の電車」を運転したことはないからです。

だけど，もしかしたら本物とは全然違っているのかもしれないんだけど，「電車でGo！」というゲームをプレイして重量感あふれる挙動を見て「そっかぁ，本物も重いんだなぁ」と思えてしまう。つまり，この「電車でGo！」というゲームは，その挙動が，想像力が「このくらい」と決めたリアリティにちゃんと沿うものになっているんです。説得力，というか……ゲームに納得させられるものがあるんですよね。

想像力との勝負

でも，いったいなぜ「電車でGo！」をプレイする人は「電車は重いから，急に止まれない」というのが納得できるんでありましょうか。

私が思うに，それは端的にいうと，人間というのは「新しく直面した出来事に驚きがあっても，それを過去の経験の似たようなものに当てはめて"そういうものなんだなぁ"と納得できるようになっている」からなんだと思います。「これは電車なんだ」と思い込めることと，「電車は重いんだ」というのをちゃんとアタマの中で連想して思い出すことができるから，なんですよね。

皆さん，電車を持ちあげたことはないですよね。でも，電車は大きいから重いはずだ，というのはわかっています。長年の経験により（いや，アタマで慣性の法則ってものを勉強してきたせいかもしれないけど），「重いものは急に止まれない」ということを覚えているからなんでしょう。たとえば，ボウリングの球を投げようとして球に引きずられてしまった経験とかもあるでしょう。ちなみに，この慣性を受ける経験を利用したギャグとして，ボールを後ろに振ってそのまま後ろに放ってしまうヤツというやつが存在するわけです（危険だし，いいかげんそのパターンは飽きたのでやめてほしいんです。心当たりのある方は気をつけましょう）。

さて，そうやって考えると，万人が納得できる「ゲーム」を作るにはどうしたらいいんでしょう。ふむ，個人個人で多少受け止め方は違っても，慣性の法則とか，物理法則自体は普遍ですから，シミュレーションなんかは物理法則にちゃんと沿っていればある程度まではみんな納得してくれるわけで，「厳密に現実をシミュレートしたシミュレーションはいけるかもしれない」という仮説が成り立ちます。

ところが，この方法だと絶対に，最終的に「完全な」ゲームというのは存在しえないんですよね。なぜなら，現実世界で起こっているすべての要素を完全にピックアップするのは無理だし，コンピュータパワーは有限ですから，そんなに膨大な計算は不可能だからです。あなた原子1個1個を数えられますか？　つまり，物理法則をそのまんま100%シミュレートすることはできない以上，「それっぽい」ものは作れても「それと同じに見える」現象を再現するのは不可能なのです。筆者，おばかさんなので（物理はいつも30点でした。ホント，よく学校を卒業できたもんだよな）難しいことはわかりませんけど，きっとそうです。

だったら，「リアル」なゲームを作るにはどうすればいいのか。考えてみてください。あまり情報量が多すぎると処理しきれない。それは人間にとっても同じです。だって，物理的に考えてみてくださいよ。いくらコンピュータに勝るとはいっても，人間の脳みそなんてたかだか千数百グラムの蛋白質でしかないんですよ。情報処理能力に限界がないわけはないんです。だから，なんらかの情報処理の単純化が人間の頭の中でも行われているはずなんです。つまりはこの辺がカギですね。

だますなら上手にだまして♥

ここで宣言してしまいましょう。リアルってのは，本当のリアルと，そうでないリアルがあるのです。もちろん，本当のリアル，というのは物理現象そのもののことです。

で，もう一方。こいつは人間が感じられる「リアリティ」です。

ある人間にとって，その人間が「リアル」と感じ取るものは，実はリアルそのものではありません。というのも，物理的に起こっていることの情報をそのまんま脳みそに蓄積したんではあまりに膨大だったり，あるいはショッキングなことがあるので整理しきれないんですね。だから，人間の頭の中では，目で見たモノや耳で聞いたことをいろいろにフィルタリングしています。

たとえば，あるところに，木の枝に大きな大きな林檎の木があったとしましょう。林檎の大きさは直径28.456786cmありました。色は#FF2738で表現できる色でした。枝の先の部分は実はナナフシですが，色は木の枝とは8%ほどコントラストに違いがあるだけです。木の枝には葉が2枚ついています。葉の色は（以下略）……でした。大きな林檎は結局落ちてしまいました。ここに書いた文章が本当に起きたことだとしたら，ここに書いた文章の内容は本当のリアルです。ところがこれを読んだ読者にとって重要なのは「大きな赤い林檎が落ちちゃった」ことだけなのかもしれません。そうであれば，「大きな赤い林檎が落ちちゃった」ことを読んだ人が理解して思い浮かんだことが，「主観的なリアル」です。

もし，この物事を再現するのであれば，現実をそのまんまなぞることができるのであれば，それもいいでしょうが，それは必ずしも必要ではありません。読んだ人の頭の中に「大きな赤い林檎が落ちちゃった」ことを読んだ人が理解して思い浮かんだことを再現させればいいのです。つまり，ゲームにとって必要なリアルとは「物理的なリアル」に沿ったものではなくて，「主観的リアル」に沿っていればいいんですね。逆に，作ったものが，あるいは「主観的リアル」から外れてしまうと「これはすべったね」ということになってしまうわけです。

さて，この頭の中のフィルタですが，ではどんなものがフィルタになるんでしょうね。筆者はそれが，「これまでの経験」だったり，あるいは「こうなったらいいな」という願望などであり，フィルタリングして単純化しているんではないかと思っています（ついでに，だいたい「経験」というのはさらにそれまでの経験だったり，あるいは大きな出来事なんかが記憶に残っているものなので，本当の現実よりは特徴的な部分だけ大きく感じることになります）。

それができないと，今度は頭の中の感情をコントロールする部分がパニックに陥ったり，情緒不安定になるんじゃないでしょうか。世の中には計算されたパニックや情緒不安定を楽しむエンタテインメントもあるけど（というか，エンタテインメントにおいて，シナリオなんかは，たとえば情緒の不安定と安心を次々とつなげていくためのプログラム，という役割を持っているんだけど），ゲームの背景にはある程度安心感を持たせるべきだろうと思います。

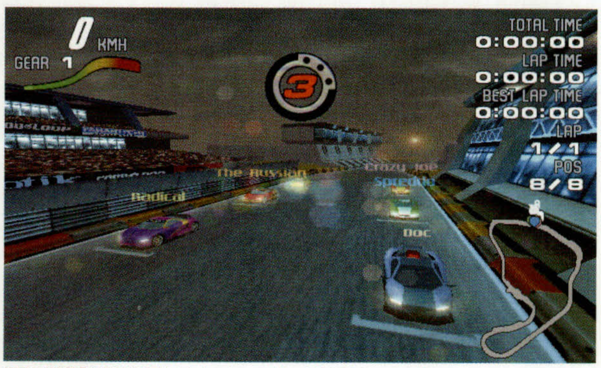
そのためには，まず，プレイヤーにとって納得できるゲームの背景を作り上げることが必要です。それは「主観的にリアルである」とプレイヤーが感じられるように，作られるべきなんじゃないかなぁ，と思うわけです。

人間は利己的なもんです。ですから，「主観的リアル」は実に適当に都合のよいことを考えているもんです。だけど，いわゆる「シミュレータ系」のゲームのもの足りなかった部分はここを満足させればずいぶん違ってくるんではないかと思うのです。ということで，ちょっと筆者の主観的リアル感をつかまれてしまったゲームを紹介しましょう。ちょっと昔のゲームになりますが，「MOTOR HEAD」というゲームです。

MOTORHEADとあかり

「あかり」といっても神岸あかりでなければ，1/1あかりハウスでもございません（またありふれたネタをやってしまった……）。この，「MOTOR HEAD」というカーレースゲーム，割と珍しいことにゲームの舞台が現代のサーキットでも街道でもなく（すでに死語感いっぱいだが）サイバーパンクな近未来だったんですね。

で，レースとしての完成度はまぁいいとして（私個人的にはかなり好きなんですけど，まぁ，PCゲーマーにはちと難易度低すぎかな，って気



筆者ご推薦「主観的リアル」なカーレースゲーム「MOTORHEAD」

がしなくもないです）ぜひ，見ていただきたいのはこのゲームのグラフィックなのです。近未来感を漂わせるためなのか，画面にはフォグがかかっていて，そして，その霞に曇る街の中で車体のハロゲンライト，街路の水銀灯が光っていて，この光り方というのがまた独特。

まー，やたらと「あかり」系のオブジェクトがハデなんでありますな。どれもこれも，やたらリフレクション，つまり反射光やら，レンズフレアが発生しているのです。スタート地点なんか見てくださいよ。ここのように，ライトが集中している部分ではそこの部分が白くなってしまって，なにがあるかすらよく見えない，という凄さなのです。すごくあかりが「明るそうだな～」という感じがしませんか，これ？　ちなみに，この画面効果には，このMOTOR HEAD独特の「DI'S SOFTWARE RENDERING」を使用してまして，それの効果のようなのです。つまり，オプションで3DFx製ドライバやDirect3Dドライバを選択することもできるんですが，そちらを選択すると，それはそれなりにあかりがあるんですが，ちゃんと車がいるのがわかる程度の明るさなのです。

でね，実をいうと，私は現在実在する車のレース，たとえばル・マンだ，なんていうのは実は見に行ったことがないんです。だから実のところ，よくあるレースゲームに出るような，車のライトもコースの街灯もただ点灯して風景が見える程度に明るい，というのが現実的なのかどうかわからないんですよね。

ただ，街道を走っていると，なにを勘違いしたかハロゲンフォグライトの黄色い光をビカビカに照らした，車種は知らんが（でもランエボとかが多いと思う）GranTourismo好きのにーちゃんが乗ってそうな車とすれちがって，あまりの強烈な明かりに視神経をやられてしまうんじゃなかろーかと思うほどまぶしかったことが二度や三度ではなかったのです。だもんで思いました。「この明るさはツボにはまってる」と。あのハロゲンヘッドライトは雨や霧が降っていてさえまぶしくてかなわんのです。フォグがかかって（ちなみにこのゲーム，フォグのかかり具合はオプションで選べます），その向こうに車のテールランプや街灯が浮かんで見えるこのゲームは，そういう意味ではまさしくツボなのでありました。

ついでにいうと，私にとって，近代的な建築物のある身近な場所といえば「羽田の飛行場」だったりするのだけど（空ちゃんだし），ここって実に夜になるってーと，夜に力いっぱい逆らうような水銀灯やらネオンやらでビッカビカで，これを連想してしまった，というのもあります（経験からくる納得，ですね）。とゆーわけで，私にとってはこのMOTORHEADが近未来のカーレースである＝霧に逆らうようにどいつもこいつもあかりビッカビカというのはとっても「主観的にリアル」で感動してしまったのでありました。「私もそれは主観的に近いかもしれんぞ」と思った方はぜひぜひ見ていただきたい，と思う１本なのでありました（といいつつ，このゲーム，あんまりヒットしなかったところを見ると私に近い主観の人は少なかったんだろーか？）。

翻ってフライトシムの話

さて，話はフライトシミュレータの話に戻りますが，フライトシムってどうなんでしょうね？　私，まだまだフライトシムには「客観的リアル」はあっても「主観的リアル」に訴えるようなゲームが足りないなぁ，と思うのですよ。

たとえば，先ほどの「あかり」の話ですが，そもそも空港にとって「あかり」ってとても重要なものなんですよね。なぜなら，だいたい飛行機というのは（視界のない計器飛行でもない限り），滑走路の照明を目指して飛ぶんですよね。空港に行ってみればわかると思いますが，夜ともなれば，飛行場は，場所によっては，周りから浮き上がって見えるくらい明るいのは確かなのですが，それ以上に，乗客として乗っていてさえ飛行機上では，光こそが心のよりどころとなるのです（翼よ，あれがパリの灯だ）。

つまり，経験上重要なものですから，飛行場のあかりというのは実際に使われているものより，心の中で相当「あれは明るいはずだ」というイメージができあがっているのです。「主観的にリアルな空港のあかり」として，やっぱり目が痛いくらいのMOTORHEADのようにぴっかぴかなものがほしいですよね。特に雨や曇りの夜には飛行機の窓から見ても空港のあかりってのは強烈なくらい光っているものだと思います。

そうそう，現状のフライトシム系ゲームで不満に思うのはもうひとつ，エアバンドです。エアバンドというのは簡単にいうと航空無線のことです。管制塔（タワー）やコントロールが空域を管理し，航空機はこのタワーやコントロールからの無線に従って指定のルートを飛行するわけです。ガリガリいう無線機からの声と英語でのやり取りは結構カッコいいので当然，主観的リアルな飛行機にもきっちり必要なものですね。というか，現時点でも，私が遊んでいる「フライトシミュレータ2000」をはじめとして，「ビギナーズパック2000」，「JETでGo！」なんかでもフライト中に聞こえてきます（あ，「ぼくは航空管制官」なんかはそのまんまですね）。

そうそう，ゲームの世界のみならず，たとえば，テレビアニメなんかだと私の記憶にある中では「新世紀エヴァンゲリオン」でエヴァ参号機を輸送する飛行機とコントロールとの交信なんていうのもありましたね。それだけ「航空」という心のイメージにはこのエアバンドは大事なものなのです。

さて，このエアバンドですが，実際の航空管制ではただの飾りなんかではなく，ルールもあるし，もちろん意味のある交信をしています。しかし，この辺がどうもリアルになっていない気がするんですよね。

たとえば「JL126, Tokyo Control. Turn left heading 090desend and maintain 3000」であれば「日本航空126便，こちら東京コントロール。左に曲がり進路を090，3000フィートに降下し高度を維持せよ」となるわけ。ちなみに，無線による管制は飛行機がゲートを離れてから，タキシング，滑走路での離陸，途中経路，そして，アプローチ，着陸，再びゲートに着くまでを管理している。つまり飛行場でこれを聞けば飛行機がどの方角からいつ着くのか，また飛行機が滑走路に着陸してからもどこに駐機するのかがすべて把握できるんですよね。

ま，それはともかくとして，基本的に発音は簡単な英語の発音になっています。たとえば進路の「090」は「ぜろきゅうぜろ」ではなく「ゼロナイナーゼロ」。滑走路の「34R」は「サーティフォーアール」ではなく「ツリーフォーライト」。発音上まぎらわしいと事故に直結する可能性もあるので，発音は一意図では一意であって，しかもわかりやす



MOTORHEADが妙にかっこいいのは，このあかりの過剰な演出なのだな。race1でレンダリングエンジンから専用に作ってしまった，このぴかぴかな演出を見よ。後続の車は光の洪水におぼれてなにも見えん。まじで



いように多少日常会話とは変えている発音もあるのです。

さて，ことほど左様に覚えれば間違えようがなくて，主観的リアリティには大事なものであるにもかかわらず，よく聞いていると意外とミスがあるのです。たとえば，「ビギナーズパック2000」をはじめとするトワイライトエキスプレス製品ではほとんどがきちんと使われているのですが，1箇所「伊丹-羽田チェッキングフライト」で090を「ゼロナインゼロ」と発音している部分があります。マイクロソフトのフライトシミュレータは日本語の音声が滑走路の「34R」のRを「アール」と教官がいっている部分があったりします。それ自体は間違いじゃないんですけど，教官がちゃんと意味のあることをいってくれなきゃいかんだろう……と思います（ちなみにエアバンド自体はちゃんとしてるし，英語版ではチュートリアルも間違いない用語を使っているのだけど……日本語訳の問題なのかなぁ）。些細なことなんですが，こういうのに気がつくと冷めちゃうもとなんですよね。どうにかなるといいんですが……。

もっとも困ったちゃんなのは「JetでGo！」。着陸後，聞き取りにくいけど「……to Taxi Way（タクシーウェイへ）」といっています。ところがこの無線の指示に従ってランウェイ（滑走路）を外れてタクシーウェイに入ると「Course Out」，つまりコースを外れたということで減点されてしまうのです。まったく，滑走路で止めるつもりなんだったら着陸したときに，そのまま滑走路に待機させておいてくれればいいじゃないですか（現実的にはあまりないが，もちろん，管制官は飛行機に滑走路に待機するように指示することはできる）。エアバンドは飾りじゃないのですから，内容はちゃんとゲームの進行にあったものを入れておいてほしいと思います。

次世代のゲームでは，ほんと，この辺は間違いなくやっていてほしいところです。

あーあ，でもそんなことをいっていると，そして，また次のゲームが出たら，それを買ってしまって遊んでしまって，「ここはああだ，あれはこうだ」と思ってしまうのでしょう。ほんとキリがないですな。

でも，「HANEDA2000」のような物量にモノをいわせてリアルさを追求しているものもあります し（私はプレイしていないのですが，これは衛星写真によるテクスチャとオブジェクトで作り上げた超精密な羽田空港のシナリーで，飛行機も羽田空港を離発着する日本の全便が現実の時刻どおりに動きます。

ホームページを見る限りでは空港ターミナルの道路標識まで本物そっくり……ちゃんと「環八通り」って書いてあるのが見える），空港の管制に関していえば，離着陸する動くオブジェクトを自動生成して視覚と聴覚（エアバンド）で日本の空の過密ぶりを表現する「リアルエアポートトラフィック」（どちらもトワイライトエクスプレス社より発売のマイクロソフトフライトシミュレータ用。「HANEDA2000」はフライトシミュレータ98用。「リアルエアポートトラフィック」はフライトシミュレータ2000用）も発売されるのかもしれませんから，十分な主観リアルも，あとわずかのところまできているのかもしれませんけどね（本当かなぁ……）。

参考文献・データ
ヒトはどうやって形を認識しているのか
http://www.sw.nec.co.jp/con/sci/1a/index.html



これはトワイライトエクスプレス社のマイクロソフトのフライトシミュレータ2000用のアドベンチャーシナリオ「ビギナーズパック」。つまり風景の見え方はフライトシミュレータ2000そのまんまである。が，やっぱりこと日本の航路に関してはこっちが断然よし。ちなみに風景は夕方の東京国際空港（羽田）だ



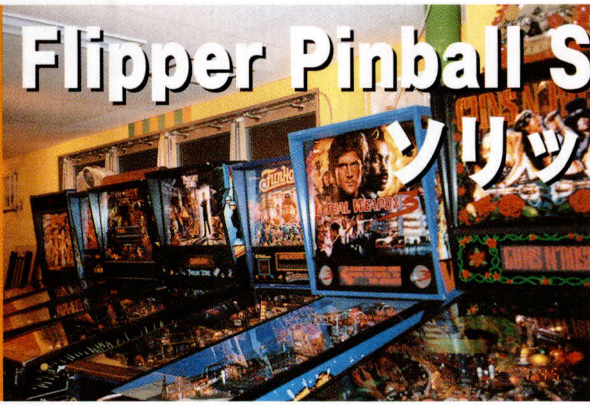
プレイステーションの「JETでGo!」。まぁ，PCのフライトシムと比較されるのは不幸としても，やはり管制塔の指示が変なのは気になってしかたがない


こっちは新東京国際空港（成田）。悪くはないんだけど，主観リアル的にはもう少しトラフィックとあかりがやっぱりほしいんだよなぁ……



そういえば，「JETでGo!」のコックピットは省略されているのはしかたがない……としても，ゲームのステージが上がると計器上に進入コースが表示されなくなるなど，旅客機とはあるまじき行為に出てくるのはちょっとどうにかならんのだろうか。「事故」という想定ならわからないでもないけど，それならそう書けば？　と思う

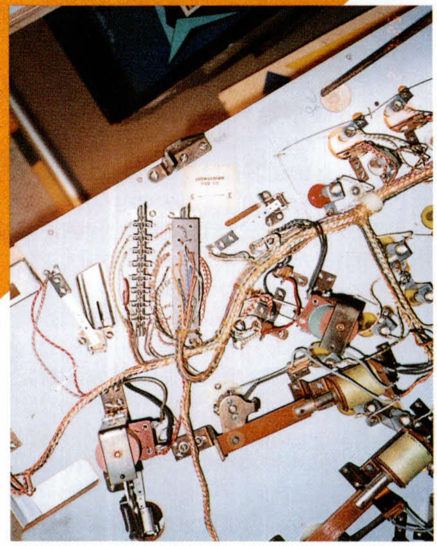
Flipper Pinball Stories #4
ソリッドステート



市川幹人　Ichikawa Mikito

現在27台のピンボールが稼働している新小岩オモロン（http://www.mindware-corp.com/mindware/pinball/location/omoron/omoron.html

Electro Mechanical（電気機械式，以下EM）からSolid State（半導体式）へ，'70年代後半よりピンボールはコンピュータを搭載する。それ以前の牧歌的ですらあったピンボールは一挙に近代的に複雑になる。それまでのピンボールは「新たな仕掛け」「新たなルール」の発明といった“質的変化”を主に発展してきたが，コンピュータの登場以降，「音声」「映像」を中心とした“量的変化”を主に発展していく。今回はそのなかでもコンピュータを搭載したことにより“質的変化”をもたらしたSolid State Flipper（以下SS）の登場，コンピュータとピンボールに関わりについて記す。



EM時代のピンボールの設定
リプレイ，3ボール，5ボールといった難易度の設定はフリッパー登場以前から存在した。コンピュータの登場により設定はDIPスイッチ→バッテリバックアップと進化し，より複雑な設定が可能になったが'30年代後半から'70年代後半までこのような形だった。

コンピュータの登場

コンピュータ登場以前のピンボールは，現在ソフトウェアで行っていることをすべてリレーの集合体の回路によって実現していた。

この方式は'30年代後半から'70年代後半まで使用された。コンピュータの登場はピンボール業界の勢力分布図を大きく塗りかえるほどの影響力を持っていた。コンピュータ登場前，コンピュータ登場後，Dot Matrix Display（以下，DMD）登場後の開発，製造の構成は表1のとおりだ。

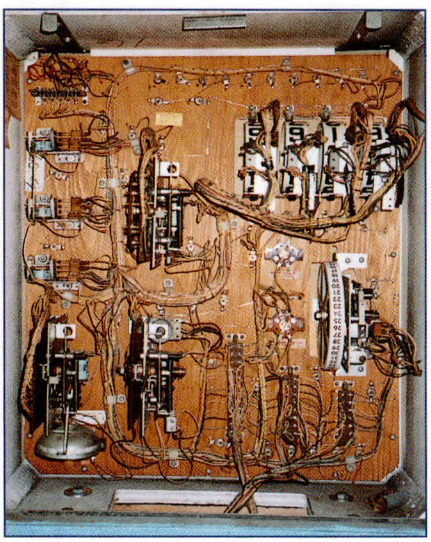
'30年代後半からコンピュータ搭載前までのピンボールの進歩は実にゆっくりとしたものだった。それでも確実に複雑化し，配線も複雑化した。配線の複雑化も限界に近づいた頃にコンピュータは登場する。コイル，スイッチ，配線の束が数枚の基板に収まり，それまでのピンボールよりも軽く，信頼性も高く，安価に生産が可能になった。

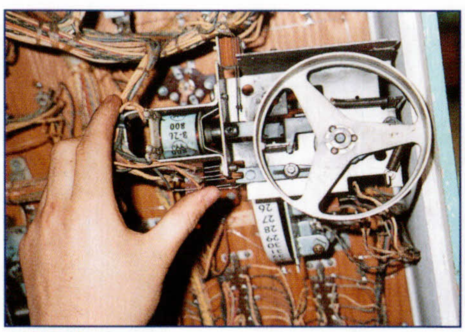
コンピュータ搭載により，より複雑なルールが可能になった。発光ダイオードによるDMDの搭載はさまざまな演出でプレイヤーを楽しませ，またプレイヤーが複雑なルールを理解するのに大変役に立った。音源の発達もまた演出面でのアピール効果，ルールを理解させるうえでのボイスによるアナウンスに使われ。ゲームに夢中になるあまりディスプレイを見れないプレイヤーに現在の状況を把握させることを可能にしている。

時代の流れで生産の多くがオートメ化されたことと，コンピュータ搭載により配線数が大幅に減ったことで生産効率は大幅に向上した。

表ではコンピュータを搭載したにも関わらず，故障率は必ずしも下がったとはいえない結果になっていることがわかるが，これはこの時期のピンボールがまだコンピュータ化の初期段階だったことが大きな原因と思われる。また'80年代中期以降においても故障率が高いのだが，これは使用ソレノイドコイル（ピンボールの動力はほとんどがソレノイドかモーターだ）が飛躍的に増大したうえに1ゲームあたりの動作回数が増えたこともが

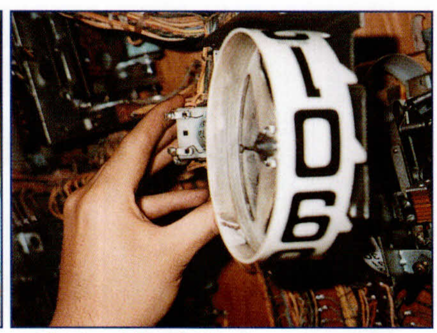
EM時代のバックボックス





10の桁のリール

4つのブレードスイッチが取り付けられている。親指側についている3本は，"0検出用"（ゲーム開始時にスコアを0にするのに使用），"9検出用"（桁上がり時に使用），パワーとなっている。コイルの右やや上方にあるスイッチは，End of Strokeと呼ばれるスイッチで，コイルが鉄芯を引くとスイッチが離れる。このスイッチの調整は大変重要で，スイッチが速く離れるとパワーがなくなり，スイッチが離れないとコイルは焼けてしまうのだ。



10の桁のリールを前から見たところ

EM時代のピンボールのバックボックス部

背面より開けるため，多くの店では（壁際に設置するため）移動しないと開けられない。右上がスコアリール。右中やや下にあるのがクレジットリール。左下は，得点が入ったときに鳴るベル。左上はリプレイの設定部分。スコアリールは10の桁（左から2番目）のみ配線が少ない。これは1の桁がナンバーマッチに，100の桁，1000の桁はリプレイの検知に使う配線があるのに対して10の桁は特に検知するものがないからだ。

重なったためと思われる。

一方，プレイフィールドの故障率は年々高くなってきている。ルールが複雑化されたため，自然とフィールドも複雑になった。特に'84年あたりを境にさかんに立体構成物が使われるようになった。坂やVUKがほとんどのマシンに搭載され，この坂を登るためにフリッパーのパワーを大幅に増大し，ボールの速度は上昇した。フィールドの大きさは"EM時代"からわずかに大きくなるにとどまったのに対して，フィールドを構成するパーツは圧倒的に増加し，ボールの速度も上昇したのだから故障率はどうしても引き上がる。

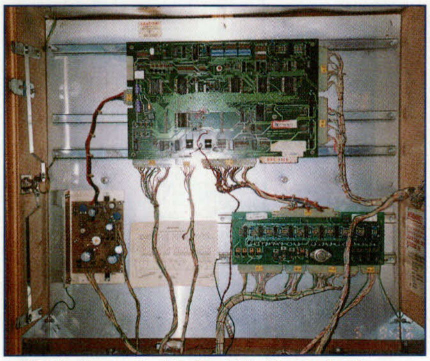
複雑なフィールドで故障が発生すると大変厄介だ。写真のTwilight Zoneなど，奥のパーツの故障が発生した場合，手前のパーツをすべて外す必要がある。また，専用パーツも増えたため，パーツの在庫切れも増え，メンテナンスは大変に厄介なものとなっていた。特に国内ではマニュアルも英文のため，最初からメンテナンスすることをあきらめている店舗が多く，フィールドが複雑になったことは国内でピンボールが激減した原因のひとつとなっている。近年のピンボールのメンテナンスは確かに楽ではない。しかし，「パーツの発注が大変」「マニュアルが英文だからメンテナンスできない」というのは理由にならない！　メンテナンスフリーは確かにひとつの理想ではあるが，ピンボールを設置している店舗のサービスマンはゲーム機取り扱いのプロなのだから，本文を読まれた遊戯場に従事する方はぜひ反省してほしい。

開発体制の変化

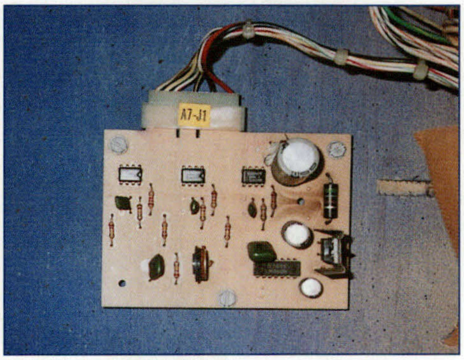
コンピュータの搭載はピンボールの開発工程自体を大幅に変えてしまった。

EM時代にはピンボール産業とは無縁だったハードウェアエンジニア，プログラマが必要となった。'80年代中盤以降はひとつの作品を作るのに必要な人数はさらに増大し，DMD搭載以降は1チーム最低で15人以上が必要となった。これは業

SS時代のバックボックス

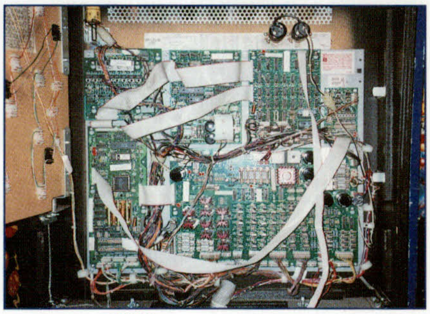


初期のピンボールソリッドステートピンボールの基板群



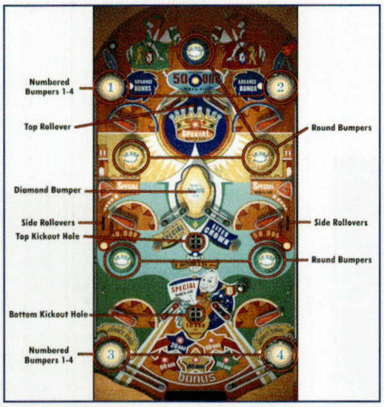
初期のピンボールのサウンドボード

ごく初期のSSピンボール，Count Downの基板群。EMピンボール時にはスコアに関るコントロール以外はフィールド下部にあったが，この時点でサウンド以外の基板はすべてバックボックス内部に格納された。また，EMピンの頃はバックボックスは後面から開けるようになっていたが，この頃から前面からバックボックスを開けるようになった。これにより，フィールドの挙動を見ながらメンテナンスが可能になりメンテナンス効率は上昇した。ボリュームつまみはサウンドボードの上に搭載されている。この後，間もなくサウンドボードもバックボックス内部に移るが，ボリュームつまみの位置は'90年代後半まで変わらない作品も存在した。この時点からCPUボード，パワードライブボード（コイルや電球の制御を行う），サウンドボードを別々の基板にする習慣があり，現在も引き継がれている。そのため，部分部分の進化に対して部分部分を変更すればよく，モデルチェンジを行っても共通部も残るため，メンテナンス効率がよくなっている。また，ピンボールにはほとんどカスタムチップは搭載されずマニュアルには回路図も付属している。日本製のビデオゲームには回路図が付属してないが，これは海賊版の出回り，ノウハウの流出を恐れたためだろう。名作ゼビウスもATARIが海外で発売した許諾版（ゲーム内容はまったく同じ）にはマニュアルに回路図が付属していて，日本と米国でのオペレータのマシンの取り扱い方の違いを感じる。
写真はGottlieb初期のSSピン，Count Down（1974/4 Ed Krinski作）。



Twilight Zoneの基板群
面積的にいよいよ限界か？　といえるほどに複雑化している。マニュアルにはこのすべての回路図が付属している。当然，このなかのコネクタの接続をひとつ間違えるだけで正常には動作しない。基板修理のためコネクタをすべて外すが，修理を終わったあとに最後の山場が残っていたりする。左上から，フリッパーコントロール＆光センサ制御基板，サウンドボード，画像ボード，拡張コイル制御ボード，左下から，CPUボード，パワードライブボード（電源，ライトコントロール，コイル制御）。ボードを徹底的に分けることにより，部分単位での進化を低コストで行うことが可能。これほどに進歩してもカスタムチップはCPUボード中央についているひとつのみ。上方やや右の電解コンデンサは，日本国内における正規代理店だったタイトーの改造によるもの。100V仕様で動作させたときに電圧低下によるシステムダウンを防止するためについている。

フィールド部－Play Field－



Humpty Dumpty（Gottlieb 1947年 Harry Mabs作）



Domino（Gottlieb，1968年 Ed Krinski作）



Count Down（Gottlieb，1979年 Ed Krinski作）



Twilight Zone（Williams，1993年 Pat Lawlor作）

Humpty Dumpty , Domino , Count Downではフィールドのパーツ点数は大きく変わらない。また，ゲームの本質としてもかなり近いものがある。Count DownはSSピンボールだがまだSSピンボール初期の作品なので信頼性の高いEMピンボールといった造りだ。仕掛けもバンパー，ターゲットレーンといったどの作品にも使われている仕掛けのみで作品固有の仕掛けは使われてない。また，立体構成もまったく使われてなかったため清掃も簡単。一方，Twilight Zoneのフィールドは大変複雑になっている。左上にあるガムボール，左中にあるミニフィールド，右上にある時計（8つのセンサを内蔵し，実際に作動する。なんとデモ中にふいに現在時刻を示すこともある）など専用パーツの盛り合わせ，コンピュータ搭載～コンピュータの進歩によってフィールドもここまで複雑になった！

界の勢力図を大きく変更した。これは主に2つの理由（段階）による。

●理由1　いままでピンボール産業に必要とされなかった人材の必要性

それまでピンボールの世界とは無縁の人材を必要とした。しかもその人材は良質の作品を作るには欠かせない。また，ゲームのプログラマの関る能力やセンスはゲームの面白さに直接影響を与える。いままでピンボールの世界とは無縁の人種だったにもかかわらず，プログラムにも長けていてピンボールにも精通している人材が急に求められた。ハードウェア面についても同様だった。ピンボールのハードウェアは大変特殊で，国内で動かすには100V-ACの電源を取るが，50V-DCのソレノイドコイルが多数ついていて，これが瞬間的にたくさん動作する。このようなハードは「この世にほかに存在しないのでは？」と思われるほど特殊だ。この部分の信頼性が低いとコンピュータ搭載の意味がなくなってしまう。

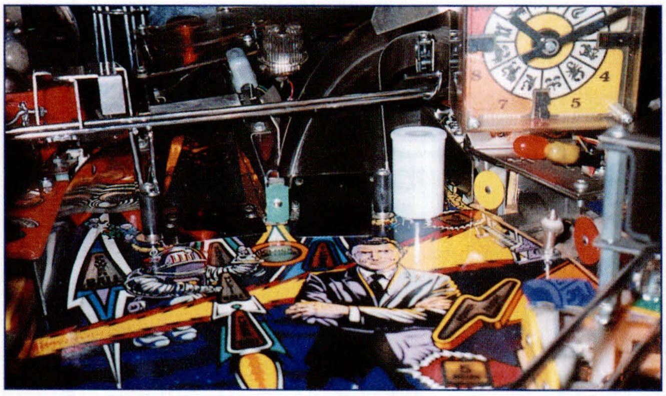
EM末期の時点ではトップメーカーだったGottlieb，Ballyはコンピュータ化に立ち遅れ（両社ともコンピュータを搭載した作品を発売したが，コンピュータ搭載を活かした作品を発表していたとはいいがたい）逆にWilliamsはコンピュータ搭載を活かした数々の作品を発表した。

特に1984年に発表されたSpace Shuttle（Barry Oursler作）は決定的な作品となり，以降1986年発表のHigh Speed（Steve Richie作），Pin＊Bot（Barry Oursler作）で確定的な地位を得る。1989年にはBally Midwayを傘下に収めるほどに業界の勢力図を変えた。

一方，EM時代には圧倒的な支持を得ていたGottliebは不振を極め，'84年には一度工場が閉鎖されるほどの事態に陥った。

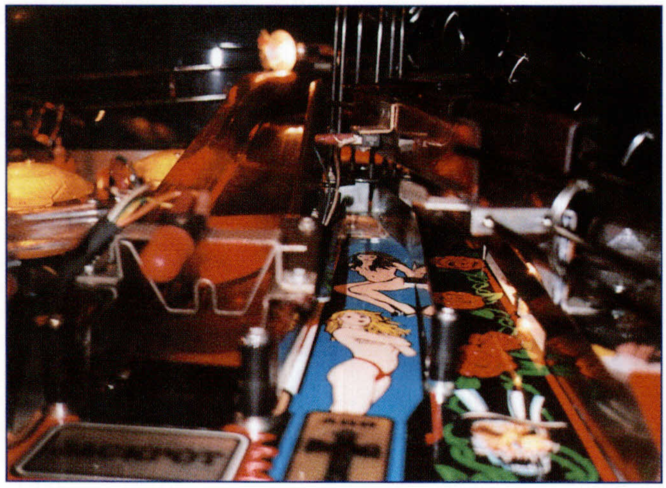
●理由2　数人での作品作りから多人数で開発する大プロジェクトへ

特にDMD登場以降の作品で顕著になるが，



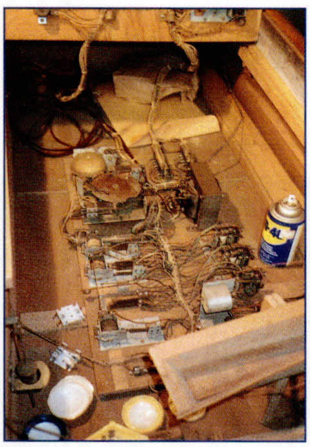
坂

坂の右にあるのはフィルムのケース。坂自体は高さ約12cm。これを登りきるためにフリッパーのパワーは増し，ボールの速度は当然速くなる。フィールドのパーツの材質，構造など改良は行われているが破損率は当然高った。球速が速くなるため，プレイヤーを選ぶ作品が増え，ピンボールが本来持っている「あらゆる人に楽しめる」という特性が失われているように思える。また，フリッパーの動きが速くなったので，"微妙な球捌きで遊ぶゲーム制"から"ボールの速さに対処するゲーム制"に変わりつつある感がある。

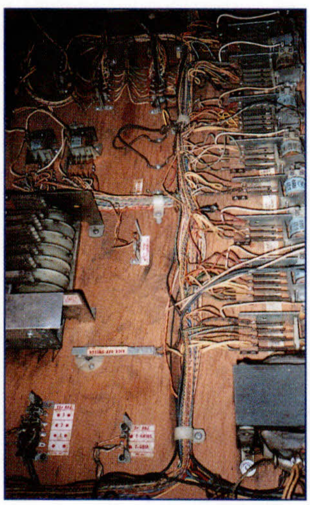


VUK（Virtical Up Kickerの略）
ボールを上方に蹴り上げる仕掛け，写真はGuns'n Roses（Data East Pinball，1994年 John Borg作）だが20cmほど上方にボールを蹴り上げる。そのため，当然信頼性は下がる。

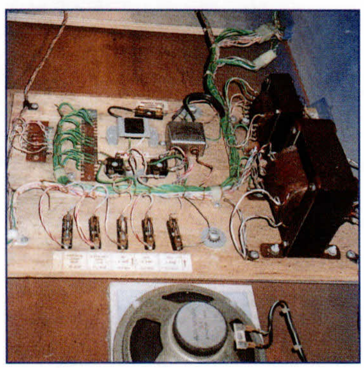
フィールド部－Under Play Field－



Humpty Dumpty
さすがにフリッパー登場最初の作品だけあってルールも単純。リレーもこの数なら扱いも簡単！



Domino
コイルとスイッチがむき出しの原始的なもの。写真は'60年代後半のものだが現在も動作する。IC，LSIと違い，規格が消滅しても修理は簡単。1000年経過してもコイルとスイッチが生きていて100V-ACの電気さえ供給されれば動作は可能。
このリレーのコイル，スイッチともに50V-ACまたは100V-ACが流れる。そのためスパークが発生し，接点の接触は確実に悪くなるのだが，この接点のうちのひとつが反応しないだけで正常な動作をしなくなる。逆にいえば，この接点さえ好調な状態なら故障は激減する。また，接点の接触というトラブルは大変原始的で目に見えるトラブルであり，また調整も容易なのも面白い。
現在のピンボールは，スイッチに光センサも使用されており，これは大変信頼性が高いが，故障した場合一般的には交換するしかない。EM時代のピンボールのスイッチは故障する確率は近代のものよりはるかに高いが，故障原因が目に見えるうえ，調整は素手でも可能。
現代のピンボールはどこかのOS同様，トラブルが起きるまでは幸せだがトラブルが起きると大変なようだ。
写真のように大変複雑な造りになるため，量産コストは大変高かった。
これがコンピュータ基板を搭載することにより，大幅にコストダウン，ワイヤーによる配線の束から基板化したことでオートメーションの製造が容易になり一挙に生産コストは減少した。



Count Down
コンピュータ搭載によって，電源部分とスピーカーのみがフィールド下になり，大変すっきりしている。
EMピンボールのときにはフィールド裏のネジやナットがリレーのスイッチに落下し，焦げたり焼けたりするという恐ろしいトラブルが発生することがあったが，フィールド下部が電源のみになったためこの類のトラブルは激減した。



Twilight Zone
さらにシンプルになった。スピーカーはこの低音用のほかにバックグラス面に2つ搭載されている。

EM時代は２人で作られた作品もあったものが，15人以上のデザインチームで開発されるようになった。さらにEM時代は３月ほどで開発されていたがDMD登場以降のピンボールは１〜２年の開発期間を必要とした。

どの世界でも同じだと思うが，デザイナーとなる人種は世間的に賛否両論分かれる類の人格の持ち主が大変に多い。いわゆる"天才"とか"変人"といわれる人づきあいのよいとはいえない人たちだ。EM時代には作品を作るのにデザイナーともうひとりくらいで十分だったので問題はなかったが，DMD以降になると15人以上。それもプログラマ，コンポーザ，各ゲーム専用の仕掛けを作るメカニック，また映画の版権モノの作品の場合，ライセンス担当の人や俳優などともつきあっていく必要があり，デザイナーには監督としての能力が重要になった。強烈な個性を持ちつつ監督としての能力も持ちあわせている人というのは，デザイナーの中でもかなり限定される。

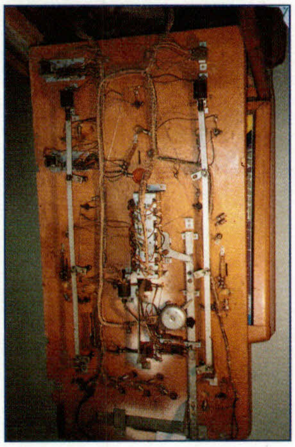
多人数のプロジェクトの時代になり，小さなピンボールメーカーは完全に消滅した。また，この時代になりWilliamsはトップメーカーの地位を確定することになる。ほかのData East Pinball（'94年〜SegaPinball，現在のStern Pinball），GottliebがEM時代に人材を足しただけの開発体制なのに対して，Williamsは完全にチーム単位のプロジェクト制を取った。特に下記の点については大変厳しく実行された。

1. デザイナーは基本的に作品単位の契約（他２社は実質給与制）

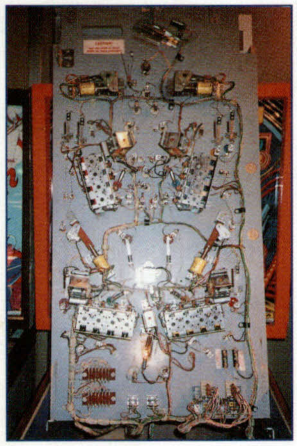
２作品続けて販売実績が悪かった場合，そのチームは解散，デザイナーは解雇された。また，「場合によってはデザイナーは会社から損害賠償を請求される可能性もある」という契約内容になっている。

日本ではここまで厳しい条件下でゲーム開発を行っている契約デザイナーはほとんどいないように思われる。作品が販売実績として成功すれば多

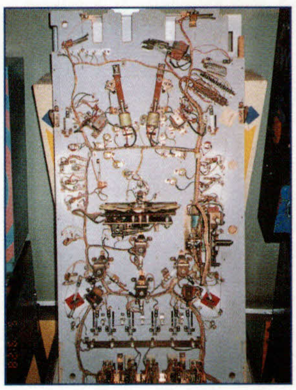
フィールド部ーfield裏ー



Humpty Dumpty
シンプル！　読者のなかには「作れそう」と思う人もいると思う。
ライトの当たるところ以外プラスチック製のパーツが一切見えないのが時代を感じさせる。
左下，右下から上方に伸びている金属板は左右のフリッパーをコントロールするためのもの。フィールド中央やや左上にあるものはベル。得点が入るときにはこれが鳴る。この時期から効果音を鳴らす発想があったのだ。フィールドの木は一枚板！　この時代は合板を作るよりも一枚板のほうが安かった。また，木材資源も豊富に存在したため，まったく問題にはならなかった。ピンボールメーカーのほとんどはシカゴに存在したが，"カナダからの木材輸入が楽"というメリットが理由のひとつかもしれない。



Count Down
かなりスッキリしている。まさに，コンピュータ化の効果覿面といったところか。
コンピュータ搭載にも関わらず，リレーで制御している部分が残っているところにEM時代の名残を感じさせる。

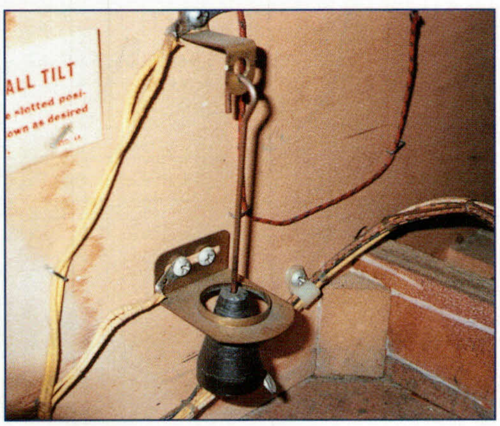


Domino
設定項目が入ったり，Roto Targetが採用されたことでフィールド裏は随分と賑やかになった。



Twilight Zone
もう，限界！　これではコストは下がらないのも理解していただけると思う。フィールド下部にある左右フリッパーのメカ（オレンジ色のコイル）が左右対称についてないのも特徴的。重力に逆らうこともあってフリッパーの装置は左右対称に設置されるのが本来は理想だ。しかし，フィールドの複雑さがそれを許さない。またこの時代になると，装置の方向の違いによるフリッパーの速さをプレイヤーが感じ取れないほどにフリッパーは速くなってしまった。近代のピンボールでは左右フリッパーが対称についている作品はほとんどない。

Misc



Tiltを検知する仕掛け。フィールドが複雑になっても，配線がスッキリしてもこの仕掛けは'40年代から変わらない。
見てのとおりというしかないが，"振り子"（Plum Bob）とそれを囲んでいる"輪"（Bracket）があり，"振り子"と"輪"が触れるとスイッチが入り，Tiltになる。この"振り子"を上方に設定するとTiltは厳しくなり，下方に設定すればTiltは甘くなる。ときおり，かなりTiltの甘い設定の店を見掛けるが，これはたいてい"振り子"を取り付けてないためである。正しい難易度の設定は，縦に突くことが可能で，横の突きに対して若干厳しく，横の揺らしに対してTiltになるのがよい。
どういうわけかこの"振り子"も圧倒的大多数の作品において左手前側に設置されている。



コインオペ業界に多大な遺産を残した故Harry Williams氏の創立したWilliamsはもう新たなピンボールを発表しないのだろうか？

くの収入を得，大失敗をすれば責任を問われる。これは，チームの結束力を高めると同時にデザインチーム全体に高いプロ意識を持たせる。

2. プロジェクトチーム同士の機密の保持を徹底する。

Williamsでは各プロジェクトチームのブースに入るには3つのセキュリティドアを通過する必要があった。デザイナーおよび，総監督の許可がない場合には上司でさえ入室が許されない。社内の人間でも作品を目にするのは作品自体が完成し，市場テスト（いわゆるロケテスト）の直前までない。

多くの開発チームを持つ会社の場合，開発が多忙になるとなし崩し的に他チームの人間を一時的に借りたりすることが起きる。また，一時的に借りることができない場合に人間関係に溝ができる可能性があるが，はっきりとチームを隔離することで原因のわからない開発の遅れを防ぐことができる。当然，開発の遅れが出た場合，原因，責任の追及は容易だ。

複数作品にわたり同じ人材でチームを構成できれば開発効率も当然引き上がる。

また，他チームの開発途中の作品を目にすることがないため，独創性のある作品が生まれる可能性が大変に高くなる。作品単位で契約するデザイナーにとって開発途中のアイデアを盗まれる心配がないことは大変重要なことだ。また開発途中にアイデアを見せると，完成の姿をイメージできない人間がいろいろと雑音を入れる可能性がある。逆に1に関連するが，デザイナーは他社の作品，他人の作品についても開発途中のものを見てはいけない契約になっている。

EM時代末期には20社近く存在したピンボールメーカーは'90年代中盤には3社になったが，この3社中でも圧倒的な1位はWilliamsで，80%近くのシェアを占めていた。残りの2社が大きく引き離されたのは，ピンボール開発が大規模なプロジェクトに変化したにも関わらず（システムとしては）前近代的な開発の仕方をしていたことが大きな原因に思える。

後書き

今回の原稿の執筆中に，'99年10月27日，世界最大のピンボールメーカーWilliamsがピンボールからの撤退，10月4日，Sega PinballがStern Pinballに名称変更を行い，ピンボールメーカーはStern Pinballのみになってしまった。

'90年代，コインオペゲームの消費速度が飛躍的に上昇した。基本的な処理速度，記憶容量，音声，画像の進化により新しい作品は以前の作品がはるかに拙いものに見えるほどの進化を求められている。しかし，開発費は増大しペイラインは高くなるばかりで商業的に成功した作品でさえ開発に携わった人が幸せになるケースは希になりつつある。

かつてのピンボールの消費速度はテレビゲームなどに比べて遥かにゆっくりしていた。しかしDMD登場以降のピンボールはそれまでのピンボールが「消費されないゲーム」という色から急激に「消費されるゲーム」に近づいたように見える。表1に〝家庭用ビデオゲーム〟の例を挙げてあるが，ピンボール，ビデオゲームともに規模が大きくなり，複雑になるに従い開発費は高騰している。テレビゲームソフトの製造費は年々安くなるがピンボールはパーツ当たりの単価は下がっても複雑なフィールドの作品が多く結果的に製造費は安くはならない。Williamsもこの問題を解決することはできなかった。

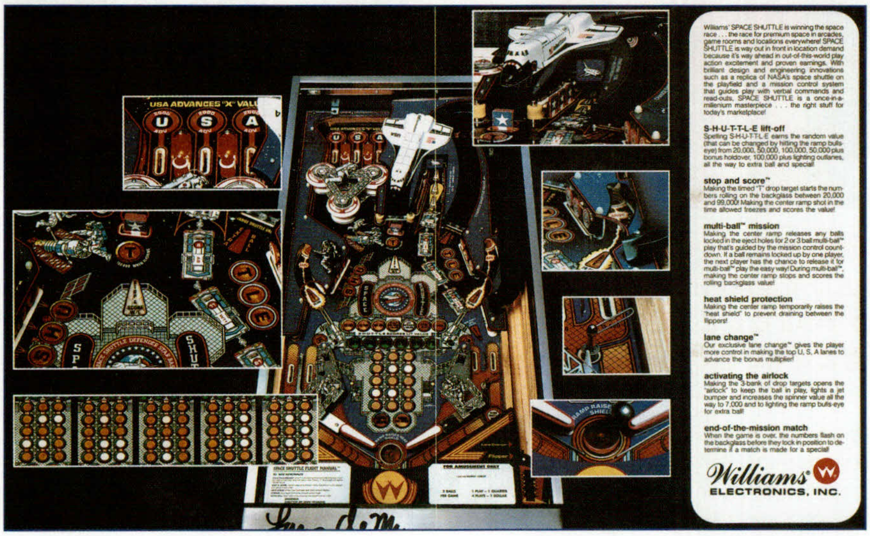
残ったStern Pinballがこの問題を解決するのか？　新たなピンボールメーカーがまったく新たなベクトルでこの問題を解決するのか？

不安を抱きつつも筆者は今日もピンボール台を直し続ける。

表1

	EM時代 ～1979頃	SS初期 1979頃～1985年	'80年代中盤以降 1986年～1990年	DMD登場以降 1991年～	近年の家庭用ビデオゲーム 1994年～
開発メンバー	デザイナー 絵描き	デザイナー 絵描き プログラマ ハードウェアエンジニア	デザイナー 絵描き プログラマ ハードウェアエンジニア メカニック コンポーザ	デザイナー 絵描き プログラマ ハードウェアエンジニア メカニック コンポーザ 画像 声優	デザイナー 絵描き プログラマ コンポーザ 画像 声優
製造効率	手作業が多い 配線が大変多い ほとんどが共通パーツ	手作業が多い 配線は大幅に減る ほとんどが共通パーツ	オートメ化が進む 配線がやや増える 専用パーツが増える	オートメ化が相当に進む かなり配線が増える さらに専用パーツが増える	メディアへの複製のみ
故障率（制御） 故障率（Field）	かなり高い 低い	高い 低い	高い やや高い	かなり低い 高い	ソフトウェアのバグがなければ ほぼ皆無
開発費	絵以外はすべてデザイナーが作るため，大変に安い。EM後期にも開発人数はほとんど変わらないのでコストは安いが，デザイナーは大変だったと思う。その面でも，後のコンピュータ化は必要と思われる。ほとんどの作品は3カ月以下で作られる	今までピンボール産業には無縁だったコンピュータエンジニアの登場により，若干コストは引き上がる。 8カ月くらいで開発は行われた	この頃から専用の仕掛けや立体構成が使われはじめる。メカニックの腕がものをいうようになる。また，ビデオゲーム人気に押されてかメーカーがかなり減った。 開発に1年かかる作品が出はじめる	いよいよ専用パーツ盛り合わせ状態。画像も音声も多大。 1作品あたり2年もかかる作品も出始め，開発費は大変に高い	非常に高い
製造費	部材は大変安く，構成も単純だが，大量生産技術は低い。時代とともに構造は若干複雑化したもののそれ以上に量産技術が向上し，若干ずつ下がる	コンピュータ面も，フィールド面もオートメ化が進んだうえにEM時代からそれほどフィールドが複雑にならなかったため，製造費は大分安くなった	専用パーツ，立体構成の登場により，製造費は安くならない	コストダウンのため，さまざまな工夫が行われるが，結果的にそれはより複雑なピンボールの登場に発展する。結果的に製造費は引き上がる傾向にあった	大変安い。 1つあたり￥500を超えるものは少ないと思われる
宣伝費	ほとんどが業界紙，または業者相手のショウのみのため，マスメディアに対しての宣伝はほぼ行われてない				大変高い
球速	遅い	遅く，牧歌的ですらある	速くなりはじめる	速い！	
特色	球速も遅く，時間制のルールもないため（ごく一部に例外あり！）がプレイヤーの立場が弱い。ルールは共通のルールが多く，そのバランスの変え方でデザイナーは勝負している面が強かった	EM時代の作品のルールに何点かコンピュータがないと実現不可能なルールを持ち込む。ユニークな作品が多かった	ハイ・スピード登場の影響か，マルチボール＆ジャックポットの作品がほとんどになった。ルールが複雑になりはじめたのもこの頃から	DMD登場により，「画面で説明すれば何でもあり！」のようになりルールはさらに複雑になる。また，ボールセーブやオートプランジャーの登場もこれを加速。フィールドも立体構成，専用パーツの多用によりさらに複雑化する。しかし，ゲームとしての完成度は抜群に高い	

Flyer



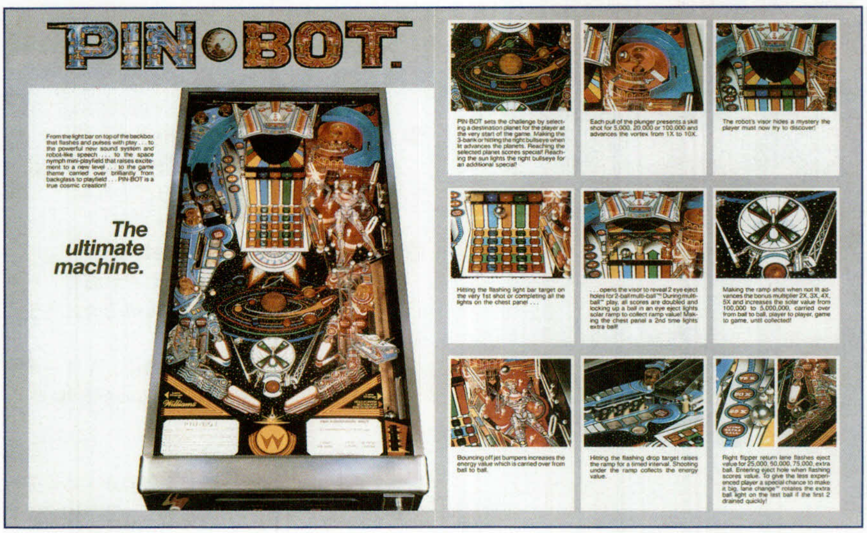
Space Shuttle
1984年11月 Williams Barry Oursler作
'81年頃にはBallyにピンボール部門が売却されるのでは？　とまでいわれたWilliams起死回生の一作！　作者のBarry Oursler氏もこの作品を機にヒットメーカーへの道を歩みだした。
以下のHPに詳しく書かれているので，こちらもぜひ参照することをすすめる。

http://members.netscapeonline.co.uk/pinballmachine/shuttle.htm

High Speed
'86年1月 Williams Steve Richie作
初のアルファベット表示可能なディスプレイ（Alpha Numeric Display）の搭載。マルチボール→ジャックポットのルールを初搭載したマシン。以降のほとんどすべてのピンボールにこのルールが搭載されたことを考えてもその影響の大きさがうかがえる。デザイナーのSteve Richie氏は，ATARIのラインワーカーからトップデザイナーになった。
以下のHPに多数の画像がある。こちらも参照することをすすめる。

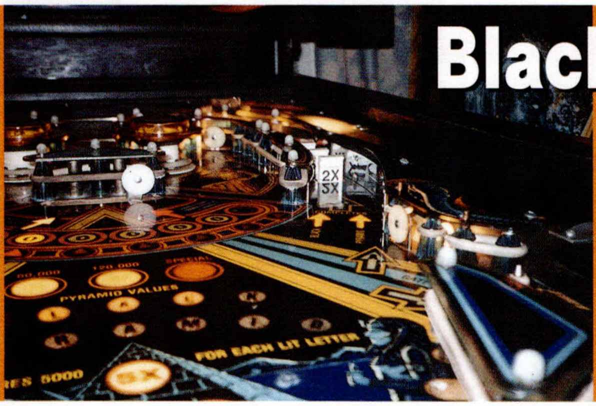
http://www.pinballarcade.com/games/pinball/hs.htm





Pin＊Bot
1986年10月 Williams Barry Oursler作
以下のHPに詳しく書かれているのでこちらも参照することをすすめる。

http://members.netscapeonline.co.uk/pinballmachine/pinbot.htm



Black Pyramid

市川幹人 Ichikawa Mikito

ピンボールの歴史については別項で扱われているとおりですが，ここでは趣向を変えて，過去の名作ピンボールを取り上げて重点的に紹介してみましょう。たくさんあるピンボール台のうち，この台のどこがどういうゲーム性を持っているのかなど，ピンボール以外でも参考になると思います

1984年，ピンボールの世界にコンピュータが導入され5年近くが経過し，コンピュータが持ち込まれる以前には不可能だった複雑なルールや時間制のルールの登場，サウンドにもPCMが使用されプレイ中の狙うポイントを声で教えてくれるようになった。プレイフィールドも複雑になり，1981年に登場したBlack Knight[*1]では2階建てフィールドも登場していた。



フィールドにルールが直接書かれているのが特徴的。インストラクションカード（いわゆる説明を書いた紙）がなくてもフィールドをよく見るとルールがすべて理解できる。　写真中央にあるのがスインギングターゲット。左右に往復運動していて時間制限役のないこの作品の中でアクション性を高めている。右側のレーンに立っている2×，3×，5×と書かれているのがDrop Target。当たると倒れ，ボーナス点の倍率を2倍，3倍，5倍にする。★200000★と逆扇状になっているライトの下の赤い三角がマーカー。スインギングターゲットにボールを当てたときにはマーカーの上の文字が点灯する。BLACK PYRAMIDの獲得した文字は写真下方の文字が点灯する。P-Y-R-A-M-I-Dの文字は獲得するとそのターゲットのライトは消える。文字は一度獲得するとすべての文字を揃えるまでは再獲得不可能。



Black Pyramidフライヤー裏

図を見るとわかるが，そんな時代に登場したこのフィールド。筆者はこの作品を見たときにいかにも工夫がないというか垢抜けないと感じた。フィールド右側のレーンにある3枚のターゲットのうち2枚がなくライトも半分近くが切れていた。それでもプレイしてみると夢中になり砂漠に吸い込まれる水のように時間は消えていった。今回はそんなBlack Pyramidについて記す。

Field

フィールドの配置は図のとおり。それぞれのターゲットに設定されたボーナスは以下のような仕様となっている。

●キックアウトホール（Kickout Hole）

入った時点でB-L-A-C-Kの点灯していた文字の内容がプレイヤーに与えられる。

- B　25,000点獲得
- L　左Bumper点滅
- A　ゲートが開く
- C　右Bumper点滅
- K　25,000点獲得

●トップレーン（Top Lane）

通過した側のバンパーを点灯させる。すでにバンパーが点滅しているときには得点のみ。

●P，Yターゲット（Target）

当たると“P”，“Y”文字をそれぞれ獲得。バンパーに当たってボールが跳ねているときにうまく獲得すると楽。

Pターゲットは右フリッパーから，Yターゲットも左フリッパーからでも狙えるのが意外。

●バンパー（Bumper）

当たるとボールを跳ね飛ばし得点が入る。通常は100点だが，点灯中は1,000点，点滅中は3,000

点になる。
バンパーに当たるとマーカーは移動する。ボールを落とすとバンパーは消灯する。

●左レーン(Left Lane)

通過すると得点が入り，以下の特典のいずれかを獲得。得点は通過回数が増えると20,000点→30,000点→40,000点→50,000点となりさらに通過するとExtra ball，さらに通過するとSpecialを獲得！　ボールを落とすと20,000点に戻される。

- B-L-A-C-Kのなかで獲得してない文字をひとつ獲得。このとき，キックアウトホールで獲得された特典はなし。あくまでも文字の点灯のみ
- 200,000の文字の中で点灯してない文字をひとつ獲得

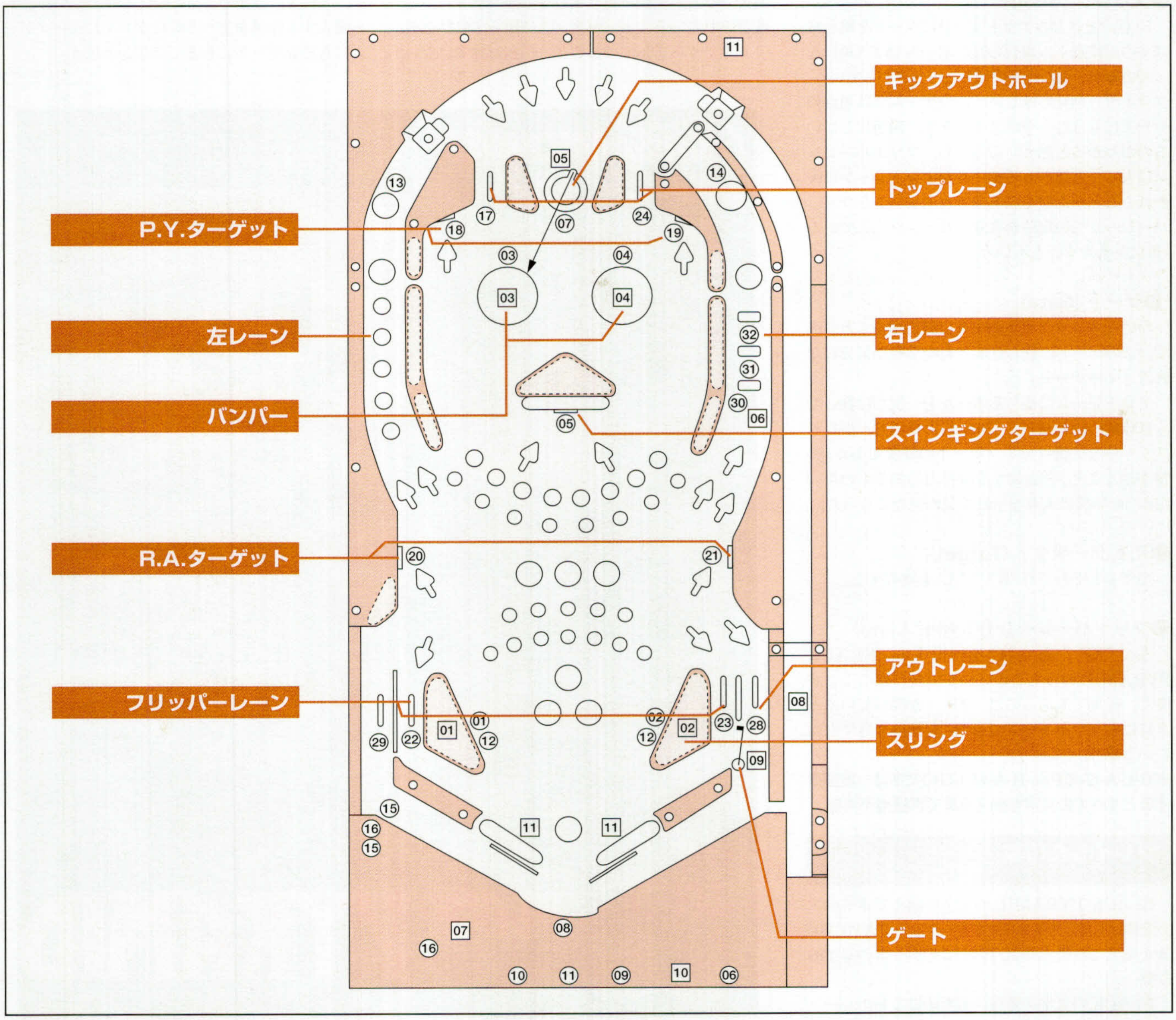
●右レーン(Right Lane：2×，3×，5×と書かれたターゲットが立っているレーン)

レーン手前のDが点灯しているときには"D"を獲得。
すべてのターゲットを倒すと，レーン手前にBLACKまたはPYRAMIDのいずれかが点灯。通過すると点灯している側の文字のすべてを獲得する。ボールを落とすとすべてのターゲットは復活する。

●スインギングターゲット(Swinging Target)

1回のショットで最大600,000点を獲得可能なターゲット。当てるとマーカーの上の★200000★の文字を獲得する。2と0の上方にある矢印ライトが点灯しているときに当てるとB-L-A-C-Kの

＊1　Black Knight 1981年Williams
初めて磁石でボールをコントロールする要素，時間制限の役，2階建てフィールドを採用した革新的な作品。コンピュータを搭載したメリットを最大限に生かした作品。Steve Richieデザイン，Larry DeMarプログラム＆デザイン。
Steve RichieはのちにMulti Ball→JackPotの流れを初めて搭載したHigh Speed，以前には初めてレーンチェンジを搭載したFire Powerを発表している。スピード感のあるゲームを作らせればピンボール史上最強のデザイナーだった。High Speed，F-14 Tomcat，Roller Games，The Get Away，Star Trek Next Generationなど，あまりにも多くの名作を残した。現在はAtari Gamesで監督として働いている。
Larry DeMarは現WilliamsのエンジニアリングMarの責任者。ビデオゲームではDefender，Robotoronといった作品が代表作。ピンボールではプログラム部分を担当しつつデザインに加わっている。'81年のBlack Knight，'88年のBanzai Run，'92年にはAddams Family，'94年にはWorld Cup Soccer '94，'95年にはJack＊Botとそれぞれの年代に名作を残している天才。



Black Pyramidプレイフィールド図

点灯していない1文字を獲得。

ここで面白いのが，★200000★は200000の文字が揃うと，そこで200,000点を獲得することだ。★は点灯していてもしてなくても獲得してしまう。★がひとつ点灯していたら400,000点，2つ点灯していたら600,000点獲得する。さらに，左レーンではこの★は点灯しないため，あと1文字で200000が完成するときでもわざと獲得せずバンパーにボールを当て★の下にマーカーを点灯させてからこのターゲットに当てる遊び方をするようになる。

また，フィールド中央にあるためターゲットにボールが当たり，そのまま真ん中にボールが落ちることも珍しくない。リスクと見返りのバランスが絶妙，で全体的に地味なこのゲームに華になる場面を与えてくれている。

●スリング (Sling)

10点，ときどきアウトレーンにボールを跳ね飛ばすので注意！　現代のピンボールでは三角になっているが，この作品ではちょっと変わっている。フライヤーを見てほしい。このゲームでは頂点のやや下にもうひとつポストを立て，四角にしているのがわかると思う。このため，フリッパーレーンに入ったと思われるボールがアウトレーンに弾かれることがたびたびありアウトレーンとフリッパーレーンでの攻防をよりスリリングにしている（単に落ちやすいともいう）。

●ゲート (Gate)

〝A″をキックアウトホールで獲得することで開く。このゲートに救われると大変幸せな気分になれるフィーチャー。

アウトレーンに落ちるボールを一度だけ救ってくれる嬉しい仕掛け。'70年代に登場し，かなりメジャーな仕掛けでピンボールの特徴である「実物が動くこと」を強調する説得力のあるものなのだが'90年代に入りまったく見かけなくなった。

●R,Aターゲット (Target)

当てるとそれぞれ〝R″，〝A″を獲得する。

●フリッパーレーン (Flipper Lane)

左で〝M″，右で〝I″を獲得する。BLACK PYRAMIDのうちこの2つが最後に残ることが多く，難関でもあるのだ。ゲートが開いているときには右アウトレーンでも〝I″を獲得できても面白いと思う。

※B-L-A-C-K P-Y-R-A-M-I-Dの文字は一度獲得するとすべての文字を揃えるまで再獲得不可能。

プレイした印象

BLACK PYRAMIDの文字を揃えてボーナス点を伸ばして，スインギングターゲットで★200,000★を揃えて高得点を狙うのがこのゲームの基本的攻略。

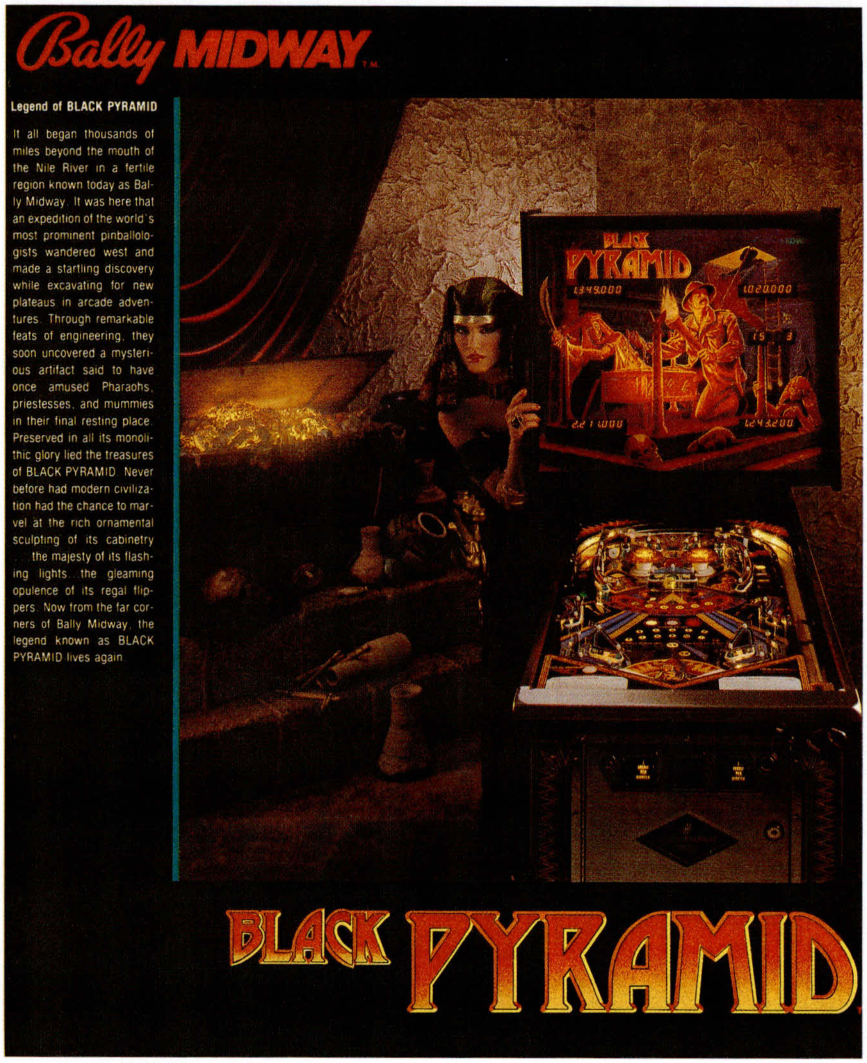
BLACKの文字を獲得するために左右のレーンを狙うが，右フリッパーでボールを打つ機会が増えれば左レーンを狙うことが多くなる。Extra Ballが近くなるとさらに左レーンを狙いたくなる。逆に左フリッパーにボールが集まれば右レーンを狙い倍率を狙うようになる。偶然，★の位置にマーカーが点灯すればスインギングターゲットを狙いたくなる。残った文字が少なくなればついカを入れ残りの文字を取りたくなる。

偶然の1ショットによってそれまで重視していなかったスポットが急激にクローズアップされる。単純な組み合わせのなか，プレイするたびに異なるゲーム展開になりプレイヤーを飽きさせない。一見簡単に感じるだけにゲームオーバーになったあとについもう一度プレイしたくなる作品だ。

最後に

それまで筆者はプレイフィールドについて，左右のフリッパーから狙うスポットがバランスよく配置されていること，ギミックに頼らずに目を引くレイアウトをしていること，左右対称ではないこと（左右対称はフィールドデザインの仕事を半分放棄しているような気がしてしょうがなかった）の3点を重視していた。しかし，目を引くレイアウトでなくても，左右対称でも十分に面白いピンボールになることをこの作品で知ることができた。フィールドを生かすルールであれば左右対称でない必要もなければ，目を引くレイアウトも不要なようだ。むしろ左右対称にフィールドを描けば左右フリッパーからの狙いという意味ではバランスは間違いなく保たれるメリットすら感じる。

近年では左右対称のフィールドの作品もシンプルなルールでその組み合わせで楽しませる作品もなくなってしまった。フィールドは複雑になりルールも複雑になってきている。ピンボールもテレビゲームもひとつのタイプの作品が隆盛になると別のタイプの作品がまったく作られなくなることが多いが大変残念に思う。一般のアミューズメント，アート，音楽のように幅広く層の厚い作品から購入する作品を選べる時代がコンピュータゲームにもピンボールにもきてほしいと思う。



Black Pyramidフライヤー表

EverQuest－妖精島探検譚



Wili@E'ci



図1　夕焼けに染まる空が美しい

あっというまにネットワークRPGのスタンダードとなったEverQuest。ネットワーク3DRPGという次期主力ゲームのスイートスポットを確実に突いてきた大作だ。次世代タイプのゲームの作り出す世界を見ていこう。

1999年春，Ultima Onlineという巨人が君臨していたネットワークゲームの世界にEverQuestは革命をもたらした。完全3Dのファンタジーワールドは UOに飽きかけていた人々をまず引きつけた。やや辛口のゲーム展開と未知のフィールドはヘビーゲーマーを中心に熱狂的な支持を獲得した。事前の次期本命の呼び声にたがわず，現在ではもっとも成功したネットワークゲームといってよいくらいの売り上げを達成している。ゲーム内容はUltimaOnlineほど多彩ではないといわれるものの，フル3DでネットワークRPGというツボを見事に突いてきた作品だ。

いずれにせよ，今後展開されるすべてのネットワークRPGにとって指標となることは間違いない。EverQuestの開発はVerantだが，SCEIのゲームということで，PS2に展開されるのかが非常に気になるところだ（HDDは必須だと思うし，メモリ足りないような気はするけど）。

次世代を先取りしたようなゲームだけに，要求スペックは高い。表示はすべて3Dであるので，なんらかの3Dビデオカードは必須。CPUも速いに越したことはない。ゾーンの切り替わりには高速なHDDがあると待ち時間が短くなるといわれている。ハードウェアに対する要求はかなり高めだ。通信回線も速いほうがもちろんよい。しかし，K6やPentiumII/300MHzあたりでも問題なく動いているみたいだ。最新ビデオカードでなくても動くし，GlideパワーでVoodoo系の信奉者も多い。まあ，その辺は好みの問題だろうが。とにかく，ちょっと前のゲーム用システムなら十分に楽しめる。

今回のシステムはPentiumIII/450MHzとELSA ErazorIII（TNT2）。もちろん，これくらいならかなり快適にゲームは進行する。

回線はモデム接続でも大丈夫らしい（当然か？）。というよりむしろプロバイダごとにかなり状況が違うとも聞く。

もうひとつの壁。英語。

チャットは重要だけど，相当にブロークンなEnglishでも通じるうえに，かなり省略した会話のほうが主体だ。さらにいえば日本人も実は多い。たまにわからない単語を出してくる人もいるので辞書は手元に置いておきたい。私自身は普通の英語のチャットルームに入ると固まるクチだが，話題の限定されているEverQuestではなんとかついていける感じ。まあ，かなり無口なほうだとは思うけど。

EverQuestとは？

ではEverQuestとはどんなゲームなのだろうか？ファンタジーRPGだというのは見るまでもなくわかる。ある意味では，ファンタジーRPGそのものである。多くの人がネットワークRPGに求めていたであろうことが実現されている。3DフィールドはVirtual Realityそのものかもしれない。

世界にはHumanやElf，Gnorm，Dwerf，Trollなどさまざまな種族が入り乱れて生活している。当然，モンスターもそのなかで息づいており，モンスターとの戦闘がゲームの主眼となって



図3　Dragnor's Cauldamの荒涼とした風景



図2　WestCommonの満月。沈みゆく月は幻想的だ



図4　パーティクルが炸裂する魔法表現

いる。世界にはPC（Player Character）とNPC（Non Player Character）がうろついており，NPC相手のQuestや，PC相手のコミュニケーションもゲームの重要な要素だが，ゲームの目的自体が戦闘に特化されていることは否めない。UOに比べて「よく戦闘だけでもたせてますよね」という人もいるけど，本来RPGってのはその部分から始まっているものだと思うし，その部分がダメだといくらイベントとかで盛り上げても所詮はニセモノだと思う。そういう意味では基本のしっかりしたゲームデザインだ。

もうひとつの特徴が3D表示に代表される，ビジュアルの充実だ。

パーティクルがビシバシの魔法表現をはじめ，岩肌のバンプマッピング，プレートアーマーの環境マッピングっぽい金属光沢など，最新技術は実に有効にファンタジーワールドの形成に貢献している。朝靄に霞むモンスターのシルエット，夕焼けの空に染まる大地など，風景の質感がとてもいい。

考えてみれば，スタンドアローンでもこれだけの3D RPGってあっただろうか？　3Dフィールドを自由に歩き回って……なんてのは誰でも考えるゲームデザインだろうが，EverQuest以前にそれっぽいものがあったっけと考えると，ほとんど思いつかない。UOの完成度とはまた違う次元で，いきなりもの凄いゲームが出てきたものだ。

シューティング系ならUnrealなどの質感の素晴らしいものがあったのだが，ダンジョンではなくフィールド型の3Dゲームというのはなぜかさほど多くないように思われる。

キャラクターの個々のモーション自体は日本の格闘ゲームレベルとはいいがたいが，モンスター系はかなりいい感じだ。

もうじき登場するといわれているEverQuest2やビジュアル強化のサービスパックにより，これらのグラフィック表示はさらに格段の進化を遂げるといわれている。「トカゲとかまんまトカゲで出てきます（kazuhisa談）」と，画像のリアリティはトップレベルのものとなりそうだ。

ゲームを始めよう

まず種族と職業などを選び，パラメータを決めて名前をつける。マニュアルを読むと，既存のストーリーのキャラクター名などは禁止となっている。甚だしいのは警告なしで削除というから徹底している。これは正しい判断だといえるだろう。世界観をウリにしていればそれを守ろうとするのは当然だ。ま，実際には変な名前もまかり通っているのだけど。

見ると，システムがファンタジーネームをジェネレートしてくれるので，せっかくだからそれを利用する。でも，呼びかけてもらうときに長すぎたり読みにくい名前はちょっと困りもの。さらにこのシステムは実に発音不可能な名前を量産してくれる。そんなこんなを考えながら何度かボタンを押すうちに出てきた，とびきり短い名前，「Wili」。

こんな名前がまだ登録されてないわけないかも，と思いつつ処理を進めるとすんなり通ってしまった（ジェネレートされた名前でも登録済みのこともあるのだ）。

パラメータは，STR（体力），STA（スタミナ），DEX（攻撃敏捷性），AGI（防御敏捷性），INT（知能：攻撃系魔法用），WIS（賢さ：防御系魔法用），CHR（カリスマ）などが用意されている。この基本パラメータはアイテムでの修正を除けば，今後一切変更できないのでかなり重要となる。種族と職業でデフォルト値は決まっており，ボーナスポイントを割り振って基本値を決めることになる。どの要素が重要かは不明の部分も多い。職業によっても違うが，マニュアルの解説は鵜呑みにしないほうがいいかもしれない。

とにかく，WoodElfのWili，Level1 Rogueの誕生だ。

やるならRogueだと決めていたのだけど，聞けばハイレベルのRogueはほとんどいないのだという。過酷な条件に行き詰まって，レベルが上がらなくなってしまうのだと。特典のひとつである毒作りの難度にも怨嗟の声が高いようだ。最近はBackStabの下方修正と苦難の道は続いている。マニュアルにもRogueを選んだ場合はほかの職業とは違うゲームになる旨が明記されている。よって，この解説もEverQuest本来のものとはちょっと違ったものになるかもしれないのだけど，とりあえず気にしないことにしよう。

Level1～3

ここは人種差別，職業差別，宗教対立の跋扈する世界。Rogueはいわゆる「ならず者」ってことで，生きていくのは楽ではない。場所によってはお店でものを売ってもらえなかったり，姿を見かけただけで惨殺される（KOS：Kill On Sight）。KOSになることがあるのはRogueだけではないけど，普通の人には安全なところでKOSになるのは辛いときもある。まあ，そういう事態に遭遇するのもまだまだ先の話だ。最初は誰でもLevel1から始まる。

WoodElfのホームタウン，Kelethinは樹の上に作られた街。Faydark（通称，妖精島）で最大の街でもある。GreaterFaydarkはマップ中に街を含んだEverQuest中で最大級のゾーンだそうだ。

樹上の街なので，当然落ちることもある。落ちると痛い，というか，たいていは死ぬ。レベルが上がってくれれば別だが，最初はまず死ぬ。

突然だが，今回のゲームシステムで使っている

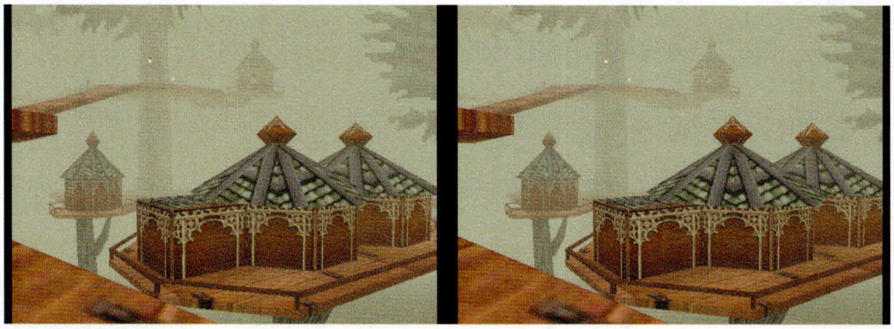


図6　絶えず霧に覆われたGreaterFaydarkのKelethinの街（平行法）

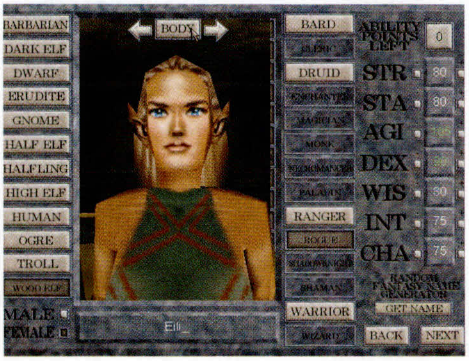


図5　キャラクターを作る。Wiliちゃんの顔はこんな感じ



図7　立体で見る蜂はこんな感じ（平行法）

ErazorIIIではRevelatorという3Dメガネが使える。そのためにErazorIIIを選んだんだから，当然このマシンにも装備されている（最近はTNT系マシン全部で使用可能だけど）。ということで，街の床の端まで寄ってそーっと下をのぞき込んでは，はしゃいでいるエルフの小娘ひとり……。ちょっと異様な光景かもしれない。

この広い世界でフル3Dだというのはそれなりに画期的なことだし，Revelatorの威力も凄い。蜂とかかなりいい感じ。目は疲れるけど。加えて，サウンドの3D効果も特筆に値する。最初はほとんどサウンドなしでやってたんだけど，最近は必ずヘッドホンを着用している。夜は暗い。種族によって夜目の利き方は違うが，WoodElfだとあたりがぼんやりとしか見えない（かなり見えるほうだけど）。そんなときの頼りが音だ。3Dサウンドなので，獲物の足音や呼吸音から方向を割り出していち早くAttackする。よい狩人には必須のスキルとなる。3Dサウンドの正しい使い方がされている例といえるだろう。

さて，前も後ろもわからない状況なので，あちこち歩き回る。初心者にしてはずいぶん変なところまで歩き回っていたかもしれない。あるとき道ですれ違ったおじさんはこちらの装備を調べて，

「なんでおまえは裸なんだ？」

なんか意味があるのかと聞いてくるが，持ってないものはしょうがないじゃない。WoodElfなんだし（それなりの鎧を着ないと画像に反映されない。初期状態ではDarkElfの次に裸に近い状態だ）。それでも親切にこれを持っていきなさいとアイテムを出してくれたので，革の手袋と布のズボンをもらった。手袋は結構値の張るものだとあとでわかった。

明らかに初心者然としているので，モンスターに襲われて逃げ惑っていると助けてもらえることも多い（最近は2nd，3rdキャラが多いので，低レベルでもよい装備を持っているキャラが実に多い）。

蜂の群れに追われていると，飛んできてくれた人がいる。こちらの体力を回復して，

「レベルはいくつだ？」

「3だけど……」

「いくら持ってる？」

「47Silverと85Copper」

正直に答える。

「これでバックパックを買いなさい」

と7Goldくれた。目茶苦茶いい人！

その後，バックパックは非常に役に立つ代物だった。思えばこの頃は銀行の機能も，お金の換算レートも知らなかった。お金は単純に10倍で次の硬貨に相当する。1Platinum（1PP）は10Gold（10GP）で，100Silver（100SP），1000Copper（1000CP）となる。硬貨は40枚で1WTの重さなので，小銭が多くなると動きが鈍くなってしまう。銀行で両替が必要になる。

Level4へ

そろそろ食料も尽きて，空腹と渇きが襲ってきていた。とはいってもレベル3までは死んだら食料とかを回復してくれるということだったので，レベル4直前にまで経験値を稼いでおいて，死体の回収の簡単そうな場所，エレベータからすぐ行ける遺跡の中で誰かが殺してくれるのを待つ。よし，これぞベストな選択。

といっても，すでにレベルが結構上がっているのでザコの攻撃だと体力がほとんど減らない。そうこうするうちにバシッと派手な音がするので振り返るとOrcちゃんが背中から斬りかかってきてる。らっき〜。この辺ではたまにしか出てこない中級モンスターだ。

気づかないふりで後ろを向いたままにしておいたけど，思うようには体力は減っていかない。結構時間がかかる。それでも，かなり減ってきたな……と思っていると，周りに青い光が舞い始めた……。これは……ヒーリング？

「Are you Ok Wili?」

振り向けば，HighElfのねーちゃんをはじめとしたパーティが救援にきてくれていた。当然，Orcちゃんはタコ殴りだ。

「I'm Okey, Thanks.」

もう，苦笑いするしかない。気がつけば盾マークの補助呪文が2つもついている。これで相当無茶なことをしないと当分は死なない。場所を変えようと歩いていたその直後，ぶつかってきたハチが勝手に死に，ピカピカピ〜ンと音がしてレベルが上がってしまった……。これも，わざと死ぬようなことは邪道だという戒めかもしれない。あとで考えたら，エレベーターの上から飛び降りれば確実に安全な場所で死ねたのに……。

初めてのPlatinum

BatWingsやBornChipsを集めて魔法使いに売るというのが初心者の金儲け法だ。そういう事情がわかってくるのはたいていレベルが上がった頃なので，最初は儲け方の効率も悪い。所詮，初心者だし。とはいえ1個あたりの相場は1GP程度だから，Orcが狩れるようになればさほどおいしい商売ではない。敵が落としていった武器などは1GPくらいで売れるものが多いのだ。時すでに遅し。

最初のうちは敵の死体から奪取したアイテムを売りに行っても，

「8Copper」

とかお寒い限りで，1Gold（1GP）分貯めるまでの道のりはとても長かった。前述のとおり，銀行の機能を知らなかったので，小銭をじゃらじゃらさせつつ，お財布を見るたびに遙か彼方にある「Platinum」の項目を遠い夢のような目で見ていたものだ。

1CPが10円だとすると，1SPが100円，1GPが1000円，1PPは1万円ということになる。小学生のお小遣いでももっとマシだよなあ，と思いつつ日銭を稼ぎ続ける。牛乳1本60円だ。

銀行の機能がわかって，稼ぎもよくなると1PP

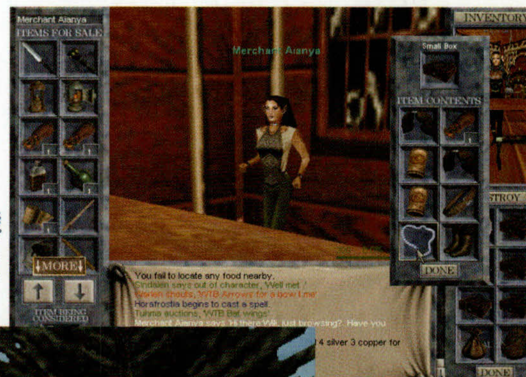


図8　ショップではアイテムの売り買いができる。ショップごとに扱うものが違うので，なにを扱ってる店か把握しておこう



図9　ホネホネ君。初心者が殺到する人気者だ



図10　キャスターによる魔法の詠唱。攻撃魔法はとても強力

も俄然現実的になってくる。そして，ついに両替をしてチャリ～ンという心地よい小銭の音，初めてのPlatinumだ。

初めてのPlatinum単位の買い物はなんだったか忘れてしまったけど，ずっと使わずに貯め込んでいた気がする。いろいろ辛い設定のゲームだけに最初の1PPの感激は忘れられない。

小径のWili

Orcリフトから Orcキャンプ，そしてCran Crushbone (CCB) エリアに至る地域はGreater Faydarkでも屈指の敵出現地帯だ。Orcリフトを降りてしばらく延びる小径，道が途切れた先のOrc集落あたりがLevel4，5くらいの初心者のよい狩り場だ。しかし途切れた道の先は何度か迷い込んで痛い目に遭っている。駆け回っていると道に迷ってとんでもないところに出てしまったりするのだ。道沿いに安全地帯まで行ける保証がないと非常に辛い。しばらくは道沿いにOrcを狩ることにした。

リフトにはガードがいるので，危なくなったら逃げ込めばいい。ガードは非常に強いのでOrcなどは一撃で蒸発する。しかし，せっかくの獲物なので，ガードに逃げ込めるぎりぎりの場所で踏ん張って戦うのがよい。危ないときにはリフトで休息している人が助けてくれることも多い。

攻撃は背後からのほうが効果的で，防御時に背を向けると大ダメージを食らってしまう。本当に逃げるときは後ろを向いて駆け出さないといけないけど，それで大丈夫な地点まではバックステップで下がりつつ戦闘を行う。

が，当てにしすぎてもいけない。ガードが守ってくれない！　Orcを引き連れてガードの前までようやくたどり着いた……と思ったら，ガードが後ろを向いてタターっと駆け去っていく……なんで？　かくして，ガードの定位置で息絶えるという羽目になった。復活したあと，ガードのところにつかつかと歩み寄って，エイと斬りかかる。当然，一撃で殺されるけど，そうでもしないと気が済まない。この税金泥棒め。

さまざまな人々

●変な人

やや慣れた狩り場でも大勢に囲まれると命を落とすこともある。

「不覚！」

ガードのところまで行き着く前にSTANを食らって倒れてしまった。ちょっと見通しが甘かった。

道の真ん中ですぐに回収できるのが不幸中の幸い。復活して速攻で平原をショートカットしつつ件のエレベータに向かう。

問題なく死体回収する。すると側にいた男が話しかけてきた。

「また会ったね」

話を聞いてみると，こいつは以前，森の中で，

「エスコートしてもいいですか？」

とかいっときながら，あらぬ方向にひとり駆け出していった間抜け男だ。

「君の死体を見張っていたんだよ」

おいおいストーカーじゃん，これじゃ。ほっといても荒らされるわけでもなし，と思いつつもWiliちゃんは無邪気でかわいい娘なので，

「まあ，とても親切な人ね！」

と，受け流す。しばらく言葉少なく当たり障りのない会話を続ける。どうやらパーティを組みたがっているみたいだが，よくわかんな～い風情でやりすごす。それでもめげないようだ。

「さようなら，My Lady！」

私のナイト1号は颯爽と駆け去っていった。

●助っ人

やはり無理をすると死ぬことはある。ちょっと小径の先まで……などと欲目を出すと，道に迷って袋叩きだ。元の道に戻ろうとしたところで道を見失ってしまい，逃げ惑っているうちにどこかわからないところで死んでしまった。死体を捜索に行くが，さらに奥地のOrcの集落の先で力尽きる。これは道の上だからまだ捜索しやすいはず。なにしろ，復活はするものの，無手丸腰なのでOrcに見つかると非常にヤバい。素手で殴って殺せるようなら，とっくに絶滅させている。もう逃げるしかない。ガードのいる場所まで逃げきれればよし，さもなくば死が待っている。死体のあるのはOrcの集落の近くで，正確な場所は不明となると，ちょっと辛い。

あちこちを走り抜けつつ探すが死体が見つからない。これまでもあちこちで襲われ，全マップ中で2番目に広いというステージを端から端まで走り抜けている。死ぬたびに死体が転がるのだが，死体には番号がついている。それが，どんどん増えていき，

Wili's Corpse (0)

なんか嫌な予感がする。プレイ時間で48時間は死体が保存されると聞いていたのだけど，まさか，1周しちゃったんじゃ……。

しかたないので助っ人を頼む。前にUOのレビューをやってたBard Kazuhisaだ。BardやNecromancerは死体を発見するスキルを持っている。

Orcが襲ってくるが，助っ人はBardといえど無茶苦茶レベル上がってるのでこのあたりのモンスターはまったく敵ではない。

この辺に2つ死体があるはずだけど，ひとつしか見つからない。しかも肝心なほうがない。その間もOrcはひっきりなしに襲ってくる。

「なんか武器はないんスか？」

「いまLootしたとこ」

Orcの死体から武器も出てきたのだ。Piercing系のRastyRapearはかなり理想的。が，部落の間近だけあって，さらに2体のOrcが襲ってくる。



図12　綺麗なチョウチョ……と思ったら胴体はドラゴンだった

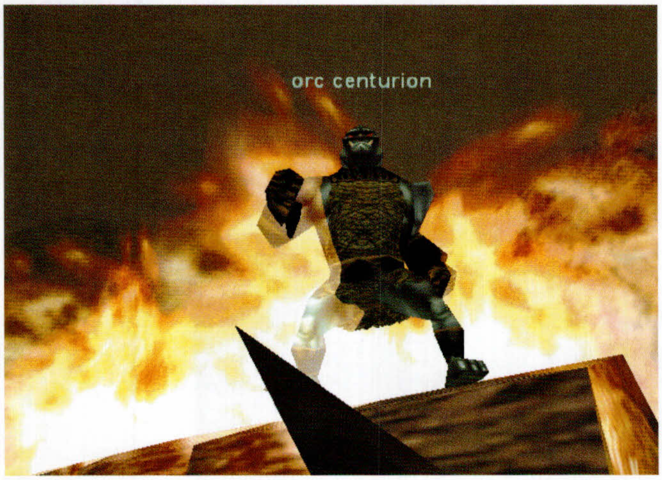


図11　最初のうちは手ごわい敵となるOrcCenturion



図13　パーティでの戦闘のようす。フレームソードは実にかっこいいなあ

「服とかはありますか？」
「いまLootしたのに入ってた」
「)」
冗談のような手際のよさで，結局，槍も手に入れ，装備をなくす前とは見劣りするけど，一応戦えるだけのものは揃ってしまった。お金はほとんど消えてしまったけど，代わりに20Platinumももらったので収支は黒字。

●変な人その2

いつものように狩りに出ていると，挨拶してくる人がいる。
「この間は君を守れなかった……」
いつのことだぁ？　誰こいつ？　と思っても，
「ありがとう。気にしないで」
と如才なく返すがよく見るとスペルミスがあるな。ち，しくった。
あまり英語がたどたどしいと日本人だとばれるなあ，日本人プレイヤーってどっちかというと嫌われてるだろうしなあ……。なんとかせねば。
お，なんかくれるっつーぞ。もらえるもんはもらっとけ。おー，指輪なんてカッコつけちゃって，このにーちゃんは。
「Merci. Thank you」
よーし，これでフランス人だ。たどたどしい英語は勘弁してもらおう。
彼が去ったあと戦利品を調べると，AGI+2のリングだった。ええもんや。さらによく見るとCarnedian Wedding Ringとある。結婚指輪ぁ？　なんか凄く早まってしまったのかもしれない。ま，いいか。

丘の上のWili

ずいぶん長いこと小径のWiliちゃんをやってたのだけど，小径でワンダリングモンスターを相手にしているだけではなかなか埒があかないので，ついにOrcの集落に行く決意をする。でも道沿いでないと迷って逃げられないので，帰り道をよく確認しつつ，丘の上に登る。
丘の上にはプレイヤーが集結しており，丘の麓に点在するOrcのテントを皆で攻撃する感じだ。ワンダリングOrcも多く，なるほどよい狩り場ではある。道や地形も丘の上にさえ戻ってしまえばしっかり把握できる。
この頃にはとっくに生き延びるための最良手は「群れを作らないこと」だと把握している。敵が1体ずつなら負けることはほとんどない。だが，2体同時にこられると，何倍も強くなる。3体だといわずもがなだ。さらにたくさんの敵に襲われて逃げ惑っているさまをTrainという。列車のように群れを引き連れて逃げる奴に巻き込まれるとひとたまりもない。が，わざわざTrainを作る奴もいる。あちこち走り回ってOrcを仲間がいる場所まで引き寄せる。Pullという戦術だ。しかし，せっかくのOrcを捌ききらずにガードに消させてしまうような間抜けも多くて困る。こういうPullingは馬鹿しかやらないもんだと思っていい。が，馬鹿は多いのだ。
で戦術を考える。群れているOrcから端の奴をサクっと叩いて，タタタっとバックステップで引き離す。で，攻撃してタイミングを見計らってさらにバックステップ。ほかのOrcの視界外で戦うのが基本だ。
武器にはDamage値と同時にAttackDelayというのが設定されている。次の攻撃ができるようになるまでの時間がこのDelayだ。Damage3，Delay21の武器とDamage6，Delay42の武器はだいたい同じ威力とされているようだ。厳密には違うのだけど。んで，攻撃したあとは待ち時間が発生する。これは殴られるだけの時間なので，攻撃圏外に退避しておこうというわけだ。離れれば勝手に追ってくるので次の攻撃タイミングには敵はちゃんと射程内に入っている。理屈はともかく，効果のほうは多少疑問があったものの（相手のリーチは思いのほか長い），やんないよりはマシと思ってひたすら励む。1匹ずつおびき出しては静かに殺し回るさまはいかにもRogueっぽい感じ。そしてOrcを端から削って回る。
Rogueには必殺技がある。後ろからえいやと斬りつけるBackstabだ。うまく当たれば通常攻撃の3倍くらいの威力がある。
Backstabができるようになると，近寄っても全然襲ってこないことがわかってきたので，堂々と背面に回り込んで斬りつけるようになる。昔は歩いていると襲われたものだが，成長というべきだろうか。ほかの人を見ていると，突然襲われたり，1匹を殴るとほかのまでやってきたりということがあるようだ。職業によっては敵を引きつけやすいともいう。いまはHide＆Sneakを使うまでもない。
こうなると安定して効率よく狩りができる。下手なパーティがやるより倍の速さでOrcを片付けていく。だいたい，馬鹿なキャスター（魔法使い）のいるパーティではまずRoot（敵を動けなくする）しようとするので質が悪い。ある程度攻撃すれば敵は逃げ出すのだが，背中を向けた敵は途端に脆いものになる。Rootされた敵は逃げ出せないので徹底抗戦してくる。パーティプレイでは時間と体力の無駄でしかない。
かくして，丘の周りのWiliちゃんは非常に速やかにOrcキャンプを掃討し，ワンダリングモンスターを片付ける。さながら，殺戮マシーン，森の悪魔の様相。おかげで丘の上に群れている人々にはほとんど余禄が回っていかないけど，臆病者に報酬がないのはしかたないよね。
普通，EverQuestでもっとも使われる単語は，
「Thanks」
だと思うんだけど，この時期かなり使ったのが，
「Dont disturb!」
結構ヤな女だったかも。

Skillを鍛える

とあるショップで見かけたDwarvenRingmail Tunicという名の鎧は，目を疑うようなスペック。
「ほしい！」
でも130PPという値段だと80PPも稼がないと買えない代物だった。その時点で1日5PPくらいは稼げるようになっている。2週間くらい頑張れば手が届く計算だ。
目標ができれば日々の戦いにも張りあいができる，というようなことを詳しい者に話すと，
「あれはイイっスよ。かなり後半まで使えますし。持ってるの使ってませんけど，いります？」
うっ。
もらってしまった。PC間のオークションでの取り引き値はショップで売っているのよりかなり安いけど，この装備は分相応とはいいがたいかも。
ともかく，防御力が上がって安定して敵を倒せ



図14　森深くにあるフェアリーの街

るようになると，ほかのスキルで鍛えてないものがあるのが気になってくる。高いところから飛び降りれるSafeFallやSwimming，SenseTrapはかなり鍛えてあるんだけど，Rogueの特権であるPickPocketが全然だ。これは鍛えるのに相手が必要なのだ。PC相手にやるとやばいし，ということでOrcPawnを適当に殴り，あとはほっといてみる。敵の攻撃はほとんど当たらないか，ダメージを食ってもすぐに回復する程度でしかない。よーしよーし。あとはご対面してひたすらPick Pocketを仕掛ける。2時間弱でLevel40まで上げる。たまに襲われているのと勘違いして助けようとする人がいて困るのだけど，「邪魔しないで」とお断りを入れる。

手助けは美徳ではあるものの，相手のゲージを確認するか，助けを求められるまでは助けないのが基本。余計なお世話をする人には余計なお世話返しで，相手の獲物を横取りしてよーく教えてあげることにしている。とはいえ，ちゃんと助けを呼ばないと，この人逃げてるんだろうか，Pullingしてるんだろうかと悩むこともある。なんとなく，しばかれてやばそうに見えるけど……という場合は一応，

「Need Help?」

とか，

「Are U ok?」

と聞いてみる。助けてくれというのなら遠慮なくバシバシ叩けばいい。

そしてCCBへ

そうこうするうちに，丘の周りではレベルが上がらなくなってきた。Orcの総本山であるCCBに入るときがきたようだ。きついのだけど，強い敵を相手にするとスキルが面白いように上がっていく。

マウスで相手を右クリックすると相手の状態を見ることができる。このときの文字の色が赤だったら，絶対に先頭は避けるべし。緑だったら余裕でOKだけど，倒しても経験値が入らないことが多い。青だと比較的よい相手。黒だとまったく互角，下手をすると危ない。黄色だとギャンブル，キャラの基本性能にもよるがWiliちゃんだと黄色には勝てない。

青い相手やイーブンの相手を見つけて狩ることが効率のよい経験値稼ぎとなる。丘時代には知らなかったのでかなり無駄をしてしまっていたなあ……と思う。

橋の上のWili

CCBでも，奴隷の捕らわれた作業坑（Pit）近くはよい狩り場だ。Pitを見回るOrcCenturion（Cent）とOrcSlaver，たまにOrcLegionaer（Lego）が出現する。1対1でLegoに勝てるかというのがCCBで生き延びる際の目安になる。Legoは，Level12〜16くらいまでは経験値，アイテムとも理想的なモンスターといえる。出現すると取り合いになるわけで，よい位置を縄張りにすることは重要だ。

私自身はソロプレイでもそこそこいけるが，たまに群れに襲われることもあり，いいところで命を落とすこともままあった。Lego相手だと体力ギリギリといった感じ。いざというときには助けが必要だ。いちばん多く組んだパートナーはRatqueen，HighElfのWizardだ。彼女は小径時代に初めてチームを組んで以来のパートナーで，その後の丘時代は私のレベルが先行してたので釣りあわなかったのだけど，CCBで再会して以来，毎晩のように橋の上でOrcを狩ることになる。まあ，割とよいペアだったと思う。

Wizardは攻撃魔法の権化だ。治癒魔法の類は一切持たない。肉弾戦では絶対に勝てないので，誰か敵を抑える役（Tank）が必要となる。これはWizardのみならずキャスター系一般にいえる話だ。モンスターはキャスターを真っ先に狙ってくるのだ。本来RogueはさほどTankには向いていない。しかし，一度キャスターに向かった敵を引き離すにはTaunt性能の高いBackstabが有効だ。ClericやPaladinならPlatemailが着込めるのでまだマシだが，ピュアキャスターは肉弾戦では非常に脆い。狙われたら敵の目標を逸らすことが必須となる。

RogueがペアにするならClericのほうがいいようにも思うが，結局は信頼関係の問題だ。超攻撃的デュオなので，体力が尽きたときに群れに襲われるとひとたまりもない。何度も死体を取りに2人で走ってれば絆もできてくるというものだ。相棒のマナ回復には時間がかかるので，そのあいだはザコを狩っているか，体力回復。Lego相手だと単独ではかなり消耗するので，ドッカンと1，2発叩き込んでもらえると非常に助かる。

DragoonDirk

以前，

「おっそろしくRapierが似合いますね」

といわれたこともあったけど，Rapierにはとても愛着を持っている。Piercing系の武器のなかではいちばん格好がいい。同系統の武器でいちばん性能がいいのはSpearだ。でも，性能よりも見た目を重視するのがWiliちゃんだ。

ちなみにこの頃は格好悪いのでヘッドパーツは革帽子のままだ。脚パーツを金属化するのもかなり悩んだ末に行ったものだ。金属系でないと防御能力は下がるんだけど，Chain系（RingもBandedも見た目は同じ）は美観的に耐え難い。滅多に着けられるものはないけど，調べるとPlate系の頭部パーツはさらにひどい。ちなみに，「美しさにこだわってもしょうがないんじゃない？」と思う人もいるかもしれないけど，そんなことはない。WoodElf（女）のPC（プレイヤー）相手へのカリスマ値は絶大だ。NPC相手にはCHA値がそのまま適用されるけど，ゲーム内の大半はPCだし，重要なアイテムの売買もPC相手だ。か弱そうな格好をしてるとどっと助けがやってくるし，いっぱいものももらえるし，ナンパも多い。生き残るにはとても重要なことなのだ。

ということで，美しさからRapierにこだわってたWiliちゃんだけど，そろそろ攻撃力の不足を感じ始める。聞く人聞く人「Dragoonっスよ」と，CCBでの2大アイテムのひとつDragoonDirkの入手をすすめられる（2大アイテムのもうひとつはDwarvenRingmailTunic）。が，このアイテムを持っているDVInnには4回ほど惨殺されている。「コイツいつかブッ殺すリスト」には載せているものの，まだまだ相手にできるレベルではない。飛び交うオークション情報を見ていると，話に聞いてたのよりずいぶん高いみたいだ。そこそこ金は貯まってきたものの，ぼったくられるのはイヤだ。

例によって詳しい者に，

「相場いくらくらいだろう？」

と聞くと，

「持ってますけど，あげましょうか」

とくる。

うっ。

もっといいもの使ってるんで，どうせ使わないという。もらってばかりだとなんなんで，20PP（最近だと破格かも）で買い取る。

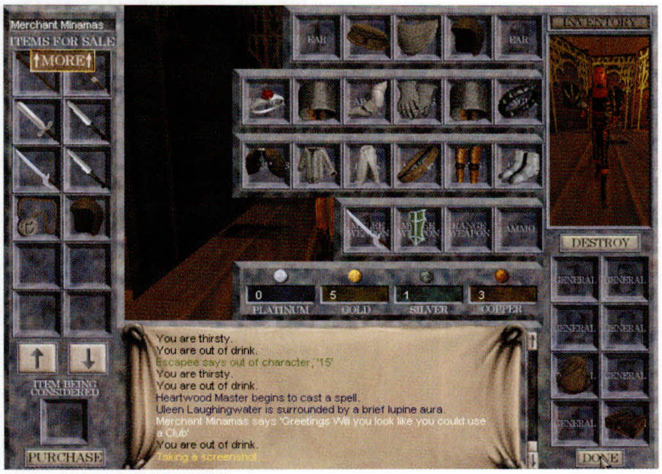


図15　この頃の装備はこんな感じ

「こいツぁ，速いっスよぉ」
といわれたけど，別段速くは感じない。だけど威力は確かに上がっているし，DEX+2の効果もついてくる。なかなかお得な一品だ（よく見るとFineSteelRapierとDelay自体は同じだった）。短剣なんで，振っている姿は少し情けないけど少しは強くなった。

そうこうするうちにDualWieled，つまり武器の両手持ちができるようになった。これもRogueの醍醐味だ。楯を使えなくなる分だけ防御力は下がるが，先日まで愛用していたFineSteelRapierを取り出して装備。またRapier使いになってひと安心。

まだ命中率は低いので一時的に弱体化してしまったものの，いずれは2倍の攻撃力が期待できる。Level16になればDoubleAttackを覚えて，さらに2倍の攻撃力だ（最初のうちは当たらないんだけど）。

しかし，防御力の低下も深刻となる。

いろいろやってるうちに，ついにヘッドパーツそのほかの導入を決める。普通はカッコ悪いとか文句をいわずに，当然レベルに相応なものを装備すべきなんだけどね。結構重かった楯を捨てたことで重量的には余裕ができた。フルBandedArmor化を目指すものの，売り手がなかなか現れない。しかたないのでChainmailCoifを購入。AC値はぐんとアップ。見た目は……まあ，今後は諦めるしかない。その後ほぼフルBanded化も達成。鍛冶のスキルを持ったプレイヤーが作った鎧は安く手に入るのでいい。このときはAC値と同じPPで購入。それ以前に使っていた装備をオークションで売ろうとしたが，買い手つかず。ショップで売ると意外なほど高く買ってくれた。高く売ったものは高く買ってくれる。結果として，持ち金は装備改善前よりかなりアップ。まあ，それだけ高値で買ってたということなんだけど。

しかし，すれ違いざまに，
「こんにちは大蜘蛛さん」
などといわれるといい気分はしない。戦闘中はともかく，街着用の着がえセットが必要かなとちょっと考える。

魅惑のイベント

巷ではにわかにLegoPadとCBbeltが高値で取り引きされている。CrushboneBeltなんてザコ倒してりゃ出てくるもので，重いだけで役には立たない。LegionearShoulderPadはLegoを倒さないと出ないんだけど，ショップではさほど高値では売れない。聞けばクエストアイテムだという。なんとなく少しため込んでいたのだけど，相方のRatqueenからいろいろ情報を集める。彼女もやったことのないクエストらしいが，必要な情報はすべて教えてくれた。

「売ると凄く儲かるわよ」
といってくれるけど（Belt 1個2PPで売り捌く女だ），いまは経験値のほうがほしい。行くべき場所はKaladimのWarriorギルドだという。最初，妖精城と勘違いしてたんだけど，CCBからはGreaterFaydarkの平原を端から端まで抜けて，BBMのゾーンを端まで抜けてと，妖精島を横断する行程が必要になっていた。

Ratqueenと別れて，とにかくひたすら走る。マップとかインターネット情報は極力使わないようにしてたけど，こういうときはマップは必須だ。とにかく走り抜ける。ようやくKaladimの城に到着，ギルドを探して，銀行を探して（とにかく人がまるでいないので困る），ようやくクエスト開始。挨拶から入って質問に答えつつ，ベルトとパッドを2個ずつ渡す。と，ファンファーレとともに経験値がぐんぐん上がっていく！　すっごい。レベルにもよるだろうが，パッド8個で1バブル，ベルトは20個（16個？）くらいで1バブルという感じだ（1バブル＝メーターのひと区切り。5バブルで1レベル）。交換でもらえるアイテムと賞金などで，1回のクエストで10PPくらいの収入もある（銀行いっぱいと手持ち分で）。実においしい。以降，2日で1 Level，40PPの収入というペースでレベルアップを繰り返す。

このイベントははっきりいってゲームバランスを大きく崩している。いろいろ聞いてみたが，おそらくEverQuest中でもっとも破格においしいQuestだ。その調整か，最近ではCCBでのアイテムの出が非常に悪くなっている。日本人のアクセスする時間帯でのアイテムの出現率がやたら低いと思うのは勘違いだろうか？　おすすめは夕方と，テレホ前の9時から10時くらいの時間帯だ。人も少なくアイテムの出現率も妙に高いような気がする。

OrcキャンプのWili

相方とのレベル差が開き，ザコでは経験値が入らなくなったのでソロプレイが多くなる。例のクエストでの経験値アップも目減りしてきたので，



図16　このおじさんにBeltとPadを渡すと経験値が上がる

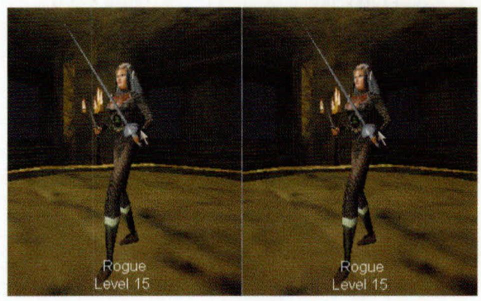


図18　この頃のWiliさん。フル装備状態だ

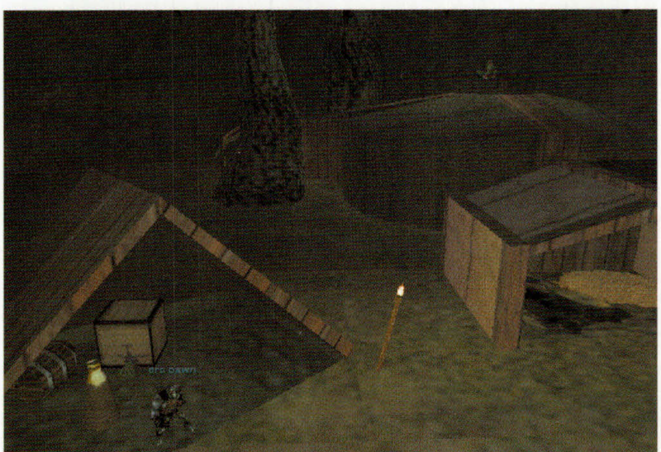

図17　この辺が最良の狩り場，OrcCamp


図19　街中では生脚に頭パーツなしの状態でいることが多い

さらに数を集めるために狩り場をCCBの奥のOrcキャンプに変更する。Orcの数が多いこと，Legoが多いこと，Trainに巻き込まれる危険がないことなど，これまでで最良の狩り場といえる。特にソロプレイには最適だ。なんやかんやいって，狩りの最大の敵はPCだ。間抜けな野郎が作ったTrainに巻き込まれなければ，そうそう死ぬことはない。

Orcキャンプのwiliちゃんは丘の上のWiliちゃんの頃と似てるけど，敵の出現タイミングを押さえちゃったのでさらに徹底している。端から殺していって，ひととおり殺すとちょっと遊びに行くか体力回復しつつ，最初の位置のOrcが出現するのを待つ。そいつを速攻で殺すと，次の奴が出てくるのでそちらに回る。出現の順番がわかっているので（というか，殺した順），先回りして出現と同時にBackstabをかける態勢を取っておける。離れたところでもいいので，目立つ場所のOrcを倒してからラウンドを開始するのがポイントだ。このトリガーOrcがSpawnしたら次のラウンドが始まるのでタイミングをきっちりあわせる。で，ぼんやりとOrcが出ないかなあと見張ってるプレイヤーたちを後目に確実にすべてのOrcを独り占めにする戦略だ（えげつないかもしれない）。

この方式はOrcの数と私の攻撃力のバランスがいいので最高の効率で稼げる。1ラウンドで11～15匹のOrcのうち，Legoが最大で5，6匹，少なくとも2匹くらいは交じっている。1ラウンド終了と2ラウンド目の開始タイミングもバッチリだ。Level18～20くらいのRogue，もしくはLego2匹同時のあとにもう1，2匹くらいなら相手できる人におすすめ。

本来なら城の中に入って戦うべき（大物はそこにいる）なのだけど，内部に入るにはパーティが必要になる。何回か入ってはみたけど，なかなかいいパーティは見つからないものだ。

EverQuestはとことんパーティプレイを中心にデザインされている。パーティでないと進めない。本来，ソロプレイはご法度なのだ。しかし，いいパーティに巡りあわなければ，ソロでやってるほうがマシでもある。Monkと並んでRogueはソロの可能性がある職業だともいわれている。やろうと思えばある程度はソロでもいけるし，リスクが大きくなる（主に馬鹿パーティに巻き込まれる）というのはあるものの，ゲームのスキルが上がれば足手纏いがいなくてやりやすい面もあるのだ。

「こんなムボーなやり方じゃやってらんないわよ。バイバイ」

といってパーティに見切りをつけたこともある。たいていの場合はかなりいい仕事をしてるつもりなんだけどね。

悪いパーティってのは，

1）リーダーが真っ先に逃げ出すパーティ（てめーは死ね）
2）Trainを作るパーティ（下手すぎ）
3）リーダーの指示を聞かないパーティ（逃げるときは逃げんだよ）

という感じで，結局のところ弱いパーティということになる。屍累々のパーティに加わってもなんにもメリットはないのだ。

いつかはベストパーティを見つけて城内を蹂躙してやろうとは思うけど，Kaladimクエストがおいしすぎてなかなかそういう気にならないのが困ったところ（そうこうやっているうちに，最近は単独で城の中に入って大ボス殺せるくらいになってしまってたり……）。

さらばFaydark

EverQuestでLevel20は特別な意味を持つ。一人前の冒険者として認められて，セカンドネームが与えられる。いわば成人式のようなものだ。話では，Level20になるとゲームマスターキャラがやってきて名前をつけてくれるということだったんだけど，それを達成したクリスマスイブの朝にはGMListには誰もいない……。その後見つけて呼んだんだけどナシの礫。しばらくたってまた見つけたGMにTellすると，

「/surnameで勝手につけて」

とやる気なしの答え。最近はそういうシステムらしい。

Level20といえば，Rogueはとうとう毒が作れるようになるレベル。毒を駆使するRogueの戦闘力はゲーム中でも屈指のものとなるという（コストは大きいし，最強はWizなんだろうけど）。とにかく今後は毒使いとして生きていくのだ。その前段階の関門の壺作りはクリアしたので，材料を必死こいて見つけて毒作りに専念するだけだ。毒を塗った刃の威力はなかなか凄いと聞く。名前は毒をテーマにしよう。

ということでつけたのが，

Acidhurricane

ファーストネームが短すぎた感じなので，少し長めに。

次はついに毒作りだ。

さて，Kelethinで毒材料は揃えたものの，毒作りの道具が足りない。聞けば，道具はなんでも揃う街，Freeportに行けば手に入るという（後日談：Freeportでは手に入らなかった……）。以前から「BBにきませんか」というお誘いもあったことだし，そろそろ故郷の妖精島を離れてもいい頃かもしれない。最大の大陸Antonicaまで海を渡って。

EverQuestではまだまだ序盤，Rogueにとっての苦難はまだ始まったばかりだ。後日談はまた機会を改めることにしたい。



図20　船だ。これで大陸へ渡れる

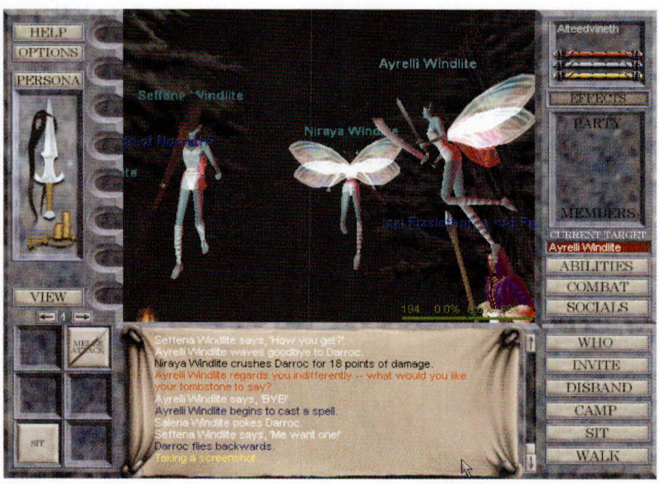


図21　GMイベント。巨大なフェアリーが出ている。このあと，突き飛ばされて，死んでしまった……

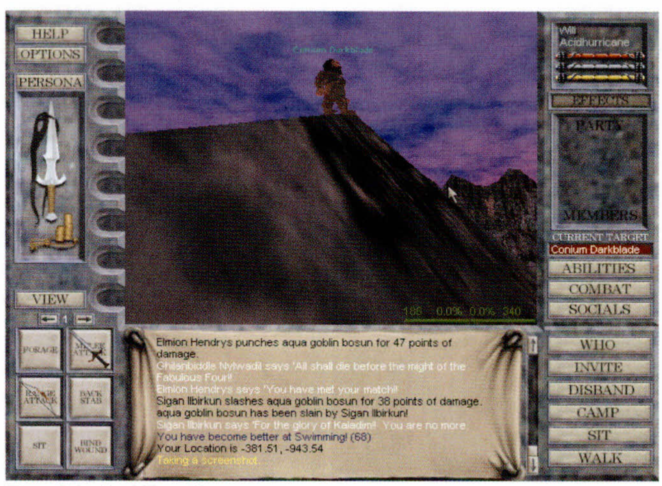


図22　頂きに立つ人影……彼はいったいなにをしているのか？

EverQuestのゲームシステムを考える

COLUMN

これだけ人気のゲームになるには，単に絵が綺麗とかもの珍しいといった要素だけでは無理である。複雑になりがちなシステムをいかに捌いているかというのを項目別にゲームシステム面から眺めてみよう。

戦闘

基本的に戦闘はオートで行われる。Auto Attack Onコマンドでターゲットした相手に攻撃を開始する。特殊な攻撃は別途コマンドで指定することになる。武器にはDelayタイムが設定されており，武器によって一定時間のインターバルが発生する。戦闘中には実行できないコマンドはあるものの，戦闘中の移動や会話などが可能なシステムとなっている。

攻撃に対してのプレイヤー側のスキルというものは反映されないか，ほとんど意味がない。一応，Auto Attackを使わないようなことも不可能ではないとは思われるが（相手の攻撃の直後などにタイミングをあわせると攻撃が入りやすい），Auto Attackで流しっぱなしというほうが普通だ。特殊攻撃を持ったキャラクターだと戦闘中の会話に加われないものの，状況把握のためのコミュニケーションは重要である。

なお，攻撃は利き手に装備されている武器で行われる。武器の種類によって必要とされるスキルは異なる。戦闘中に武器を変えるなどといったことも可能だ。

魔法

通常の魔法は一定量のマナの消費を伴う。また，魔法の使用に際してなんらかのアイテムの消費を必要とするものもある。特殊なダガーであるとかある種の宝石であるとか，強力な魔法はえてしてなんらかのアイテムを媒介するものが多い。

が，Bardの歌声などはスキルによるものだが，魔法と同様の効果をもたらすがマナは消費されない。そのほか，マナのパラメータを持たない戦士系の職業のものでもマジックアイテムを使用することで魔法の効果を得ることは可能である（とてつもなく高いので普通は使わない。魔法使いとパーティ組め，ということらしい）。

Faction

その地域や種族に対する評判のようなもの。Factionが高ければ温かく迎えられ，低すぎるとKOSとなる。

種族や職業，カリスマ値と密接な関係にあり，買い物の値段などに直接関係するほか（売ってくれないことも多い），Questで必要なキャラクターが口を聞いてくれないとかいった自体を招くこともある。BardやEnchanter，Crelicなどはカリスマが高く，RogueやDarkKnightなどEvil属性の職業ではたいていの場所で苦労する。

Factionは街の周りにいるモンスターを殺しまくると改善される。そのほか，街中のQuestをこなしていけば改善されることも多い。逆に街中で不埒な真似をするとFactionが下がるのはいうまでもない。また，どこかのFactionが上がると，それと敵対する勢力ではFactionが下がるのが普通だ。

NPCとQuest

NPCに話しかけると（Factionにもよるが），Questが与えられることがある。会話中のキーワードについて問いただしていけばよい。単語しか見ていないので，自然な会話とはいいがたいが，逆にいえば英語ができなくてもまったく問題なく会話でき，会話の流れや脈絡などもQuestには無関係だ。街角で人相の悪いBardを見つけて"I deliver kelethin"と話しかけるのが基本。

このあたりの会話システムには改善の余地は大いにあるが，前述のとおり，文法解析などをちゃんとする自然会話システムができると困る人は多いかもしれない。

生産スキル

EverQuestでは生産スキルはあまり重要視されていない。Questの一環で必要になるものもある。ギルドに行って基本スキルをつけたうえで，作り方の本などを買って，所定の道具を使って所定の材料をCombineすると製品ができあがる。失敗すると材料はおじゃんだ。当然のことながら，有用なアイテムほどレアな材料を必要とし，作るのも難しい。スキルを鍛えるには，ギルドでボーナスポイントを使ってスキルを習うか，ひたすら作り続けるしかない。

アーマーやEnchanterの能力によって作られる魔法アイテムなどは一定の需要はあるものの，積極的に必要とされるものではないということで，あくまでもついでについているものという感が強い。

ゾーン

EverQuestのゲーム中のレスポンスは非常に速い。ワールドは数十台のサーバに分かれて管理されており，それぞれのサーバ間では交流はまったくない。基本的に同じ世界があるのだが，並行世界が展開されている。キャラクターを作るときに最初にどのサーバ上に作るかを選択するのだが，ほかのサーバ上に移ることはできない。サーバ単位で独立した世界であり，常時，複数のEverQuestが動作していることになる。さらにそれらのサーバとは独立して各エリア（ゾーンという）ごとにサーバは分かれており，ゾーンへの入り口/出口でそれぞれのゾーンが接続されて全体的な世界が形成されている。ゾーンを越えるときには，それぞれのゾーンデータを読み込むので，数十秒から数分の時間を必要とする（システムに搭載されたハードディスク速度に大きく依存する。IDE RAIDが人気だ）。

それぞれのゾーンは独立して存在しており，プレイヤーキャラクターのみが移動できる。要するに，とんでもない敵に襲われても，ゾーンしてしまえば完全に逃げ切ることが可能である。これはゲームバランスに大きな影響を与えており，ゾーンの入り口あたりでは時としてモンスター密度が非常に濃いことがある（大きなトレインを作ってそのまま逃げる奴が多い）。安全地帯としてのゾーンというものがないと現在のプレイ環境は大きく変わってしまうだろう。逆に，ゾーンした瞬間敵に襲われて命からがら引き返すという展開もあるわけだが。

このゾーン別の管理という割り切った方式によって，EverQuestはパフォーマンスと拡張性，メンテナンス性などの確保を実現しているといっていいだろう。

コマンド

/（スラッシュ）の後ろにコマンドワードを書くことで各種コマンドを実行できる。

自分の位置を表示する「/LOC」などは知っていると知らないとではまったくゲーム展開が違ってくる。

また，特に効果は期待できないが，"/EM SMILE"などとすれば，感情表現も可能だ。これらのコマンドは随時追加されているので，全貌はちょっと見えない。公開されていないコマンドも使ってみれば使えたりする。間違ってもなにも起きないだけなので，いろいろやってみるといいだろう。先ほどの/LOCも本来は未公開コマンドだ（バグ報告用らしい）。

チャット/会話

キーボードからの入力は，通常は/SAYコマンドで切り換えることも可能。入力コンソールラインが出ている状態ではすべて会話として処理される。入力行の出ていない状態では登録されたコマンド処理が先になる。

Aなどはデフォルトで1キーコマンドに設定されているので，NPCに"What"などと聞こうとしてAttackしてしまい，返り討ちにされるなんてのはお約束の事故である。

会話には比較的狭い範囲でだけ有効な/SAY，エリア全域に有効な/SHOUT，/OOCなどがある。/SHOUTは赤文字で表示される緊急用メッセージ，/OOC（Say out of character）はぽつりとひとり言をいうような感じで使用される。特定の相手へのメッセージは/TELLで行われる。話しかける相手の名前を指定しなくてはならないのは面倒だが，相手からの/TELLがあれば，RキーでREPLYが使用できるので第一声があれば，あとはReplyのやり取りだけで済む。

そのほか，特殊なものとしては/AUCTIONがあり，売買関係の会話で使用される。効果は/OOCとほぼ同じだ。

/YEALは助けを求めるときに使われる専用コマンドで，声がする方向が同時に示される。

このゲームでは，パーティでの行動が基本になるのだが，パーティ内だけで有効なものが/Gによるグループチャットだ。

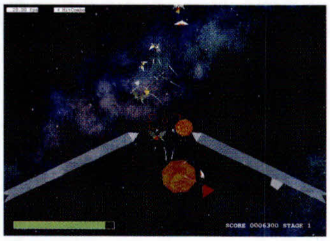
ネットワークでの多人数ゲームではプレイヤー間のコミュニケーションが非常に重要となる。特定の相手とのチャットはもちろん，使用価値の高い近くの人との会話，そして情報収集で欠かせないエリア全体的な会話を矛盾なく混在させている。さらにいえば，使用言語を切り換えて，特定の種族/職業だけに通じる会話（その言語スキルが低い人が読むと文字化け状態になる）など，会話に対するゲームデザインは非常に練り込まれているといっていいだろう。これって，専用チャットシステムより優秀じゃないか？

付録CD-ROMの使い方

付録CD-ROMは本誌記事で製作，あるいは参照したファイル群を中心にまとめられています。この付録CD-ROM(DISC1)の基本的な部分はISO9660 Level1でアクセス可能なはずです。一部，Level2を要求される部分もあります。圧縮ファイルの展開などは各自で適切なツールを用意して行ってください。

DISC1

特集関連



SION2000 (Windows)

プログラム作成　菊地功

Direct3D Retained Modeで作られた3Dシューティングゲームです。SIONシリーズをご存じの方にはお馴染みのスタイルのゲームですね。移動はカーソルキー，スペースバーでノーマルショット，Zキーでホーミングレーザーを発射します。

GALAXY-KNIGHTSの制作(前編) (Windows)

プログラム作成　サイバーヘッド

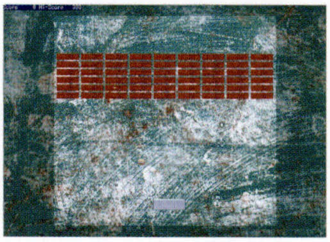
次号で完成する予定の3Dネットワークゲームの核となる3D表示とネットワーク接続のためのスケルトンプログラムです。まだゲーム本体はありません。



VisualBasicによるスプライトエンジン (Windows)

プログラム制作　中野修一

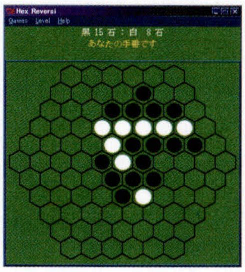
DirectX7とVisualBasicを使ったスプライトエンジンです。DX7 SDKのサンプルを見て頭を抱えている人におすすめ。



対戦型テーブルテニス (Windows)

プログラム制作　飯田伸一

DirectX(elライブラリ)を使用した典型的ブロック崩しと対戦型エアホッケーゲームです。



Tcl/Tkゲーム集 part2

プログラム制作 広井 誠

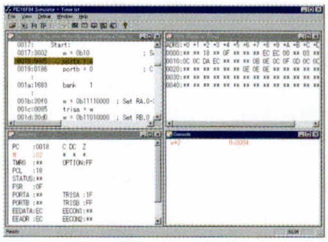
1999春夏号に引き続き，X Window，Windows，Macintoshで手軽にGUIツールを構築できるTcl/Tkを使ったゲームプログラム集です。夏号で収録したプログラムのデバッグ情報も参照してください。

特別企画/その他

R4100/R4300用ソフトウエアシミュレータ『simips』(各機種)

プログラム制作　中森章

MIPS RシリーズCPUのシミュレータです。添付オブジェクトはx86機種用(Windows)EXEファイルです。前回のアセンブラ，シミュレータに続いて，シミュレータの登場です。プレーンなCプログラムですので，基本的に機種は問いません。各自でコンパイル＆調整してください。



オリジナルPICアセンブラ＆PICシミュレータ (Windows)

プログラム制作　高尾克彦

PICマイコン用のアセンブラおよびシミュレータプログラムです。かなり実用的な完成度となりました。開発のお役に立ててください。

Your Dream, My Dream (全機種)

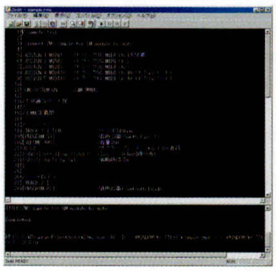
データ作成　田中順子

CGエッセイで使用されたグラフィックデータです。プレビュー画像と元画像が収録されています。元画像はかなり大きいので注意してください。



NetBSD for X68K (X68000)

X68000シリーズ用のNetBSDパッケージです。実行にはMMU搭載CPUが必要です。インストールの手順などは134ページからの手順を参照してください。



Windows版Z-MUSIC制作用ツール (Windows & X68000)

プログラム制作　わいやぎ/MASA

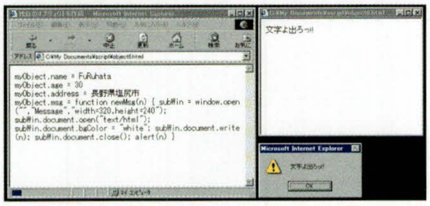
Windows環境でZ-MUSICのMMLレベルでデータ作成が可能なエディタzeditとZ-MUSICシステムver.2.0/3.0用データのスタンダードMIDIファイルコンバータなどの各種ツール群です。ZEDITは最新版(ver.0.62)と動作が安定しているver.0.5が収録されています。通常使用時はver.0.5をおすすめします。「Z-MUSICってなに?」という方はZ-MUSICシステムver.3.02のマニュアル一式をご覧ください。

Step to the Black Arts

FutureBASIC講座

プログラム制作　古旗一浩

Macintosh用FutureBasicを使用したプログラミング入門です。今回はFutureBasic ver3紹介で使用したベンチマークテストプログラムです。



JavaScript入門

プログラム制作　古旗一浩

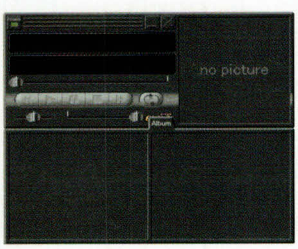
今回はJavaScriptのJSファイルを使用したプログラムの再利用とオブジェクトの扱いについて解説しています。



オルタナピコピコゲーム入門 (Windows)

プログラム制作　飯田伸一

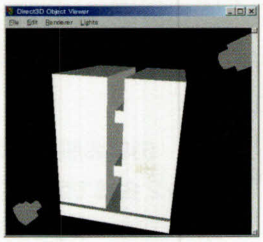
C言語から手軽にDirectXを使えるelライブラリを使用したピコピコゲーム制作入門です。前回制作したスネークゲームの完成編です。対戦が可能になりました。



VisualC++入門 (Windows)

プログラム制作　菊地功

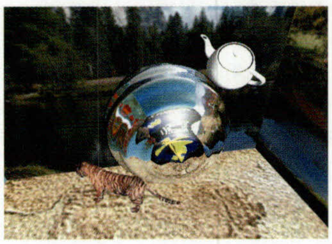
Windows Media SDKを使ったMP3Windows Media Audioのオリジナルプレイヤーを制作します。既存のプレイヤーで満足できない人はいろいろ改造してみるのがよいかも。



Direct3D RM入門 (Windows)

プログラム制作　菊地功

3Dゲーム用オブジェクト制作に適したモデリングツールフリーソフトの六角大王のデータファイルからDirect3D Xファイルへのコンバータを制作します。ポリゴンの表裏自動判定で的確なデータを出力します。



Direct3D IM入門 (Windows)

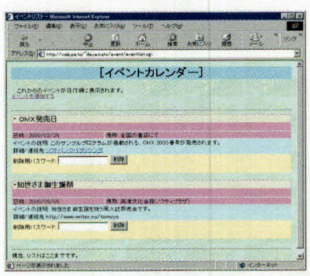
プログラム制作　菊地功

DirectX7で追加されたキューブマッピングを使って「nVidiaのデモ(これはOpenGL)のようなもの」を作成します。キューブマッピングによる映り込みとバーテックスバッファによるアニメーションです。

文字電話でWebページを見る

プログラム制作　大和哲

ちょっと特殊な分野ですが, 文字電話やH"などのメール端末でWebページを見るためのCGIスクリプトです。



iモード用CGI

プログラム制作　大和哲

最近流行りのiモード用Webページと通常のWebブラウザの両方に有効なページのためのCGIスクリプトを考えてみます。



AfterDark用モジュールの制作 (Macintosh)

プログラム制作　水野貴明

FutureBASICを使ってスクリーンセイバーソフトAfterDarkのモジュールを制作します。

X68000でCD-ROMドライブを制御する (X68000)

プログラム制作　舩本昇竜

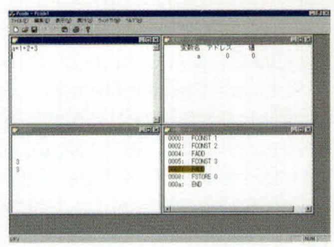
X68000からSCSI制御関数を使用してCD-ROMデータを読み出したり, 制御コマンドを送信して音楽CDの再生を実現します。



仏日翻訳ソフト来々仏語 (Windows)

プログラム制作　石上達也

フランス語翻訳ソフトがWindows対応版になりました。



言語処理系プログラムを作る (Windows)

プログラム制作　石上達也

SX-BASIC風の中間コードインタプリタ型の言語処理系を作成します。

Tools

記事制作上で使用された各種ツール群を同時に収録しています。

六角大王

制作　株式会社終作

独自のインタフェイスで左右対称な物体を作る

には最適なポリゴンエディタです。フリーソフト版六角大王（Windows版/Macintosh版）とより高機能化して製品化された六角大王Superの体験版を収録しています。

D-Pixed

制作　土井淳

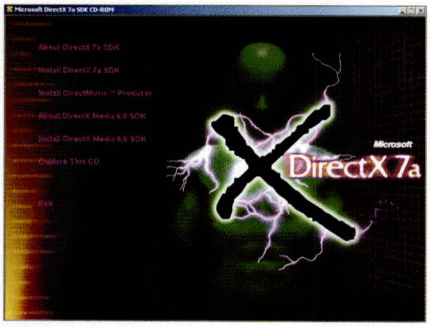
256色ベースのグラフィック制作に適したフリーソフトのグラフィックエディタです。ドット打ちほかインターネットコンテンツの制作などに有用です。

DISC2

DirectX 7.0a Software Development Kit

提供　マイクロソフト

DirectX SDKの最新版です。最新ランタイム，開発環境などを提供します。また，Direct Music Producer，Media SDKなどを含んでいます。



ライセンスおよび使用上の注意

CD-ROMに含まれるMicrosoft DirectX 7.0a Software Development Kitの権利はMicrosoft Corporationに帰属します。その使用および，これに基づく製品の開発，頒布はMicrosoft DirectX 7.0a Software Development Kitのセットアップ時に表示される使用許諾書（英文），およびMicrosoft DirectX 7.0a Software Development Kitを含むディレクトリ内DXF¥LICENCE¥REDIST.TXT（英文）の各条件に従わなくてはなりません。Microsoft DirectX 7.0a Software Development Kitの使用，または使用不能から生じるいかなる損害についてもMicrosoft Corporationは一切責任を負いません。Microsoft DirectX 7.0a Software Development Kitの内容に関するお問い合わせ，サポートは受け付けておりません。

マイクロソフト



第12回 アマチュアCGAコンテスト入選作品上映会のご案内

例年お馴染みのアマチュアCGAコンテスト発表会が関東，関西の2会場で開催されます。今年は例年より募集時期を延ばして作品募集が行われたため，発表会の時期がずれておりますのでご注意ください。

今年は応募作品の質，量とも例年をはるかに上回る作品が集まったと聞いております。発表会当日は入賞作品各種の上映のほか，ほかでは見られない謎の映像も多く発表される，非常に貴重な発表の場として知られています。CGAに興味があるという方は情報交換の場として，また「CGAってなに？」という方でも問題ありませんのでふるってご来場ください。

●開催要項

日程：東京地区 4/23（日）なかのZERO大ホール
　　　関西地区 4/29（祝）尼崎市アルカイックオクト（予定）
時間：開場12:30 開演13:00 終演17:00
内容：CGアニメーション作品約20作の上映，展示，入選作品集ビデオの販売 など
入場：無料

（写真は昨年度の入賞作品です）



第11回作品賞
「It's gauche to say.」腰原仁志



第11回映像賞
「BATTLE CHASE 2」河野達也



第11回ユーモア賞
「Mission: Li'l bit」山口秀行

特集

Game Creation!





ゲームというものが現在ではすっかり市民権を得て，老若男女誰しも娯楽のひとつとして楽しめるような世の中になっている。完全に市民権を得ただけでなく，産業としても重要な位置づけにあると見なす人が増えてきたようだ。いつのまにかゲームプログラマとかゲームデザイナといったものが小学生のあこがれの職業にラインアップされたりしているのだから世の中わからない。にもかかわらず，ゲームというもの自体は「極端に難しいもの」の代表のように扱われていないだろうか？「ゲームを作る」ということは，10年前ならプログラミングの動機の大半を占めていただろうし，その頃からやっている人ならば，ゲーム画面を見ればなんとなくどんな処理系になっているか理解できる人も少なくないだろう。最近のパソコンの性能はそこらのゲーム機に決してひけを取るものではない。最近のパソコンではプログラミングが複雑になったといわれているが，ユーザーレベルでプログラマブルであるという本質は変わらない。また，複雑なら複雑なりにそれを支援するシステムもちゃんと整備され，かなり開かれた環境にあるのも事実なのだ。WindowsのDirectXは多少の難点はあるものの，最先端ゲームを作るための環境一式がほぼ揃っているといっていいだろう。3Dは当然として，ネットワークゲームなど，相当のノウハウを積み重ねないとできないようなサービスが公開されている。使用環境も，VC++以外にDelphiなどにも対応するほか，VisualBasicが加わって裾野も広くなった。これはもう活用するしかないだろう。ゲーム制作環境はDirectXに限ったことではない。GUI用のプログラミング環境はそのままゲーム開発環境となりうる。それはWebコンテンツでも例外ではない。どんなところでもゲームは作れる。世界はゲームのためにあるのだ。

CONTENTS

ゲーム制作環境を考える

中野 修一　Nakano Shuichi

PlayStation登場でスーファミ時代のプログラマが失業し，最近ではPlayStationのプログラマでも失業者が巷にあふれているという。プログラマ受難の時代だ。「ゲームを作ること」にはさまざまな種類の人材が必要となる。そしてそれを支えるシステムが必要となる。現在，そして次の世代でのゲーム制作環境に求められていることをまとめてみたい。

ゲームというのはある意味もっと評価されてよいジャンルの表現手段だと思う。成長中の分野なので，評価はされにくいだろう。こういうのは形骸化してはじめて評価されるってのが世の中の基本だ。画家なんて死なないと絵の値段が上がらないのとも似ている。

ゲームの本質はなんだろうか？

プログラムはロジックそのものであり，ロジックならではの希求すべき目標がある。データ出力はすなわちアートであり，ビジュアル，サウンドの総合芸術であり，場合によってはドラマとしての側面を強く主張する。そして，インタラクティブ要素はそれらすべてをまとめたうえで，さらに高次元なものに置き換える。なんやかんやいって，プレイヤーというのはゲームの構成要素のなかでもっとも精妙で高価な部品なのだ。ここに高性能なパーツを使えば，ブロック崩しも芸術作品と化す。ただのボタン押し器にしているとちょっともったいない。

その最後の部品をうまく使うには……ということで，エンタテインメントであるという部分が本質といえるだろう。プレイヤーを楽しませることに力を注ぐこと，その部分をデザインしていくことがすなわちゲームを作ることだといえる。

プログラマサイドの時代

昔話とか愚痴を始めるつもりはないのだが，かつてはパソコン（マイコンか？）を扱う目的の大半は「ゲーム」だった。いまさらいうまでもないだろうが，最近ではゲームを作ることが流行らなくなっているようだ。ゲームというのがプログラマひとりでできる範囲を徐々に超えていった，あるいはそう感じさせるようになったのが大きな原因だろう。音楽作って，グラフィック描いて，プログラム作って……という人もいないではないのだが，作業量は昔に比べて格段に増えている。支援ツールなどもいろいろあるので，一概にはいえなかったり，チームを作れば解決できたりはするのだが，そんなことをしなくても巷にはゲームがあふれているというのも事実ではある。わざわざ手間をかけて作るまでもないと思う人も多いだろう。

かと思うと，作れるもんなら作ってみたいと思っている人も案外多いということがわかった。

昨今のゲームは，絵描きさんさえいれば，あとはちょちょっとしたプログラムで体裁は整う。全体の作業の大半とゲームのデキの半分はグラフィッカーの双肩にかかっているといってもいい。実際のところ，ゲームというものも定型化してきているので，そうそう斬新なアイデアによる新ジャンルなど出てくるものではない。じゃあ，バリエーションに乏しくなったかというと，そうでもない。アレンジの仕方次第で，ありふれたゲームも新鮮に生き返るのだ。

極端な話，プログラム部分は共通で，データだけを差し替えれば別ゲームになるようなシステムを作ることもできるだろうし，それは結構必要とされていることでもあると思うのだ。デザイナ主導，グラフィッカー主導ではプログラマにとってはあまり面白いことではない（そもそもプログラマにとって面白みのある部分は少ないのだ）。そのように効率化することによって，システムがある程度軌道に乗ったところでプログラマはいらなくなってしまう可能性もある。ということで，これはこれで微妙な問題を抱えているのかもしれない。しかし，ゲームデザイン優先というのはよりゲームの本質に近い立場なのだ。

アーティストサイドへ

アーティストサイドの立場で見て必要なのはプログラムの手間をできるだけ省いたシステムだ。データを変えるだけで別のゲームができるようなもの，そう，それはコンストラクションキットという形態になる。純粋にデータだけで別のゲームにできるような形態はちょっとないかもしれないが，サンプルで動いているものをそのまま別デザインにするようなことは比較的無理なくできるだろう。逆にいえば，アーティストサイドの人たちは自分の絵でゲームが作れれば満足という人も少なくないのだ。誰も見たことのないような独創的なゲーム……というものを格別に求めているわけではない。

私が見る分には，最近ゲームを作りたいという動きのもっとも活発なのはこの手のアーティストサイドの人々だ。

コンストラクションキットとか，そういった手のものではアスキーのツクールシリーズなどが昔から有名だ。RPGツクールあたりからそれらは延々と続いており，Windows上のラインアップで見ると，RPG，2D格闘，シューティング，シミュレーションRPG，恋愛ゲームといったバリエーションのほか関連ツールがいくつか発売されている。それなりのクオリティでのゲーム作成が可能で，できあがった作品の配布や市販なども可能になっている。まあ，これはこれで悪くないので，興味がある人はちょっと試してみるのもよいかもしれない。

しかし，これらは本質的に後述のデザインサイドの人のために作られたようなツールであって，豊富な添付データとか定型的なビジュアル構成などはアーティストサイドの人にはあまり喜ばれないかもしれない。

たとえば，純粋にグラフィックデータがあったとする。で，この人はコミケでCGイラスト集を売っているとしよう。売れない。で，付加価値をつけたいと思う。ゲームにしたい……という思考過程はさほど不自然ではあるまい（妙に具体的な気もするが，たぶん気のせいだ）。

こういった局面で要求されるゲームは特に複雑なものではない。ゲーム自体にはたいして意味はないのだが，ちょっとした暇つぶしになるくらいのものであればいうことはない……という感じだろう。1枚の絵がブロック崩しになって15パズルになって，ついでにデスクトップアクセサリとか時計とかに使えると非常にうれしい。馬鹿みたいに簡単な話だが，これまで壁紙にしかならなかったのと比べると格段の進化だ。こういう場合，ゲームの中身自体は完成していて，いじる要素は少なくてもいい。オーソドックスで完成度が高いものが望ましい。マニュアルを読まないとルールがわからないものよりは，誰でも知っているゲームのほうがいい。

プログラマサイドの人とアーティストサイドの人の利害は一時的にせよ，一致を見ることが考えられる。現在，こういう需要は結構あると踏んでいる。現在のところ，同人ゲームというのは，かなり低いレベルとかなり高いレベルで大きく隔たりがある。プロまがいの人もいれば，15パズルレベルの人もいて，さまざまだが裾野が広がっているという感じはない。どんどんサークル数が増えていった時代はすでに遠くなっているように思う。はっきりいえば，全体的にジリ貧傾向に陥りかかっている。

これを救うのがアーティストサイドの人々だと私は思うのだ。そして，そこに必要なのは15パズルレベルに人の作った15パズルではなくて，プロまがいの人の作った15パズルだ。バリエーションは多いけど，ある程度のレベルは保証されていることが重要だ。世の中には市販のキャラクターもののデスクトップアクセサリ集よりマシなものを作れる人はごろごろしているはずだ。そういう状況ができてこそ，裾野が広がっていくのだと思うのだ。裾野が広がっていけば，よりバリエーションを求めてジャンルも広がっていくだろう。

どちらかというとプログラマサイドの人が多いと思うOh!X読者には，今回はこういうところに目を向けてみてほしいわけだ。

デザインサイドの現場

最初にもう1回確認しておくけど，デザインというのはゲームデザインのことで，ゲームの構造などに興味を示す人のことだ。純粋にデザインサイドの人っていうのがいるのかどうかはっきりとは知らない。私が知ってるゲームデザイナの人は軽いプログラムくらいなら作っちゃうし，絵がまったく描けないというのでは話にならないからだ（普通，図くらいは作る）。とりあえず，「プログラムも作れないし，絵も描けないけど，凄いゲーム作りたいんです」系の人とはちょっと違う。

分類すると，全体的に作りたがる人と，部分的な改良を得意とする人がいるように思われる。プロデューサータイプとディレクタータイプとでもいいますか。

ツクール系のツールは一応，両者とも適合するだろう。RPGなら細部の演出にこだわる派と全体的なストーリーにこだわる派のどちらでも一応対応はできる。ゲームシステム云々にまでこだわる派にはちょっとどうしようもないが。

また，最近の話題はフロムソフトウェアのPC参入第1弾ということでまもなく発売される「Sword of Moonlight」は同社のKing's Fieldが作れるというRPG作成ツールだ。実際，サンプルとしてWindows版King's Fieldがついてくるという。これなどもコンストラクション系だが，アーティストサイド人が絡むには，3Dモデリングが必要と思われるので，どちらかといえばデザイナーサイドのツールといえるだろう（3D CGに人だとまるで見当外れだけど）。

ツクールシリーズがいわばスーファミ時代を彷彿とさせるラインアップできているのに対し，多少は近代的路線だといえるだろう。

個人的な感想で悪いが，私はフロムソフトウェアにあまりいい印象を持っていない。King's Fieldシリーズは2までは一応やったが，取らなきゃいけないアイテム（伝説の剣かなにか）を取らずにラスボス倒したためか，ストーリーとかよくわからないまま終わってしまったのを覚えている。無意味な落とし穴に落ちたり，厳重な防備を撃破して突入した部屋になにも置いてなかったり，

「ゲームってもんをわかってねーよな」

と偉そうに嘯きながらゲームをやってた気がする。いい部分もいっぱいあるし，やりたいことはわかるんだがねぇ。もっと洗練させなきゃヤバイだろ，と栄進座の舞台から北島マヤを降ろした原田菊子の心中がよくわかるような作品だった（本当にそう思った）。

そういうのが自分でゲームデザインできるようになれば少しは楽しめるかもしれない（時間があれば……）。そもそもこの手のツールは，開発者が完成したゲームを基にしたり，特定のサンプルを想定して開発してたりするものなので，「サンプルより完成度の高いものって作れるの？」とか逆に「この程度しかできないの？」いう部分で気持ちがなえさせるというのが最大の問題点だと思うのだ。すべからく，よいツールというのは機能の充実度以前に，作り手の士気を刺激するような要素が重要なのだ。

その他からの可能性

さて，こういった市販ソフト以外にもゲーム作成キットあるいはシステムのようなものはいくつか発表されている。探していけばキリがないのだろうが，いくつか目についたものを取り上げてみたい。

■JavaNovel

http://hp.vector.co.jp/
authors/VA006860/

タイトルのとおり，Javaである。

アドベンチャーゲームをWeb上で展開できるという点は新しい方向性で，Webコンテンツ側にしてみれば斬新な展開が可能なのだろうが，オンラインならではの強みというのはない。Javaなので，機種やプラットフォームをあまり問わないというのはかなりの強みではあるのだが。

■シナリオ君

冬コミで売られていた同人ソフトで「日本語でシナリオを書くだけで，ゲームが作れる！」というものがあったので買ってみたのだが（シナリオ君統合キット：http://www3.plala.or.jp/ootake/），シナリオというよりはプログラムそのものに近い。ぴゅー太っぽい感じだ。数値なども全部全角文字を使うというのが新しいのかもしれない。こっちは同人ソフト向きのパッケージだ。デモ版が配布されている。

どちらも割と似たようなタイプのアドベンチャーゲームを中心に考えたようなパッケージである。用途と目的も比較的わかりやすい。絵描きさんだけではゲーム制作は難しいかもしれないが，シナリオ屋さんと絵描きさんの2人構成であればそこそこのものは作れるだろう（どちらを使っても）。

このようなパッケージをもっと多分野で展開していくことが当座の必要事項であろう。

■StreatFighter XVI

X68000では有名な2D格闘ゲームシステムだが，アングラなところから

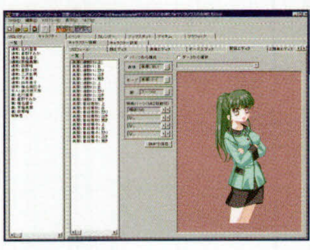


図1 恋愛シミュレーションツクール現状での最新作。ツクールシリーズでは特異な感じだ



図2 フロムソフトウェアの新作情報ページ

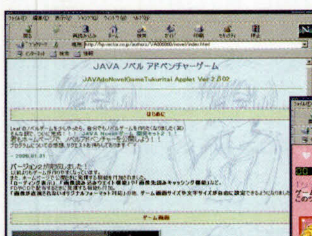


図3 JavaNovelのサポートページ。Novel形式以外のJavaゲームシステムも公開されている



図4 シナリオ君のサポートページ

発生しているものの，キャラクターの分離によってシステムそれ自体を表に出すことはできるようになった。もう少し日の目を見てもよいのかもしれないが，まだまだアングラな性格が色濃く残っているのが実情だろう。

プログラマサイドの負担がやや大きく，データを用意するのも簡単とはいいがたい。対応機種が限られるなどの問題がある。システムさえ作ればデータ自体のポーティングはさほど難しくないかもしれない。なんとなれば専用エミュレータという手もあるだろうし。

そして次世代へ

ゲームというものが人々の生活に定着しているにも関わらず，一般の人とそれを制作する側との立場は非常に隔絶した関係にある。それを埋める努力は本当に必要なことなのかというのは置いといて，これからのゲーム開発に必要なシステムというのはなにかをちょっと考えてみたい。

ゲーム環境として有力なもの，WindowsではDirectXとしてまとめられており，それはそれで必要なものを集めてはあるのだが，ハードウェアの規格化と一般化というPCならではの諸事情を呑み込んだ一大プロジェクトだ。アップグレーダブルでコンパチビリティのあるソフトウェア基礎環境が構築されている。はっきりいって，これは偉業である。

・入力機構
・出力機構
・ネットワーク機能

よくこんな面倒なことをやろうとしたものだと思う。最初から整備された環境ならともかく，PCというなんでもありな環境で通用するものを作るのは各ハードメーカーに大きな影響力を持つOSメーカーにしかできないことだ。

とはいえ，こでだけでは十分ではない。

先ほど挙げたようなゲームシステムもほとんどが2Dをベースにしたものだが，時代の流れは3Dに向かっており，その土壌で一般的なシステム構築が求められている。DirectXは3Dをサポートしているが，そのような下層構造の整備とは別のシステムはどうしても必要になってくる。

プログラマ，アーティスト，デザイナといった区分でのアプローチを適用するなら，3Dアーティストの作ったオブジェクトをリアルタイムで動かすということが前提となる。そしてこれは，多少の妥協は必要だろうが，ほぼ実現可能なところまできているといっていいだろう。

データコンバータ，ポリゴンリデューサ，ワールドマップエディタ，ビジュアライザ，オブジェクトマネージャ，イベントマネージャ……といった環境の整備でほぼ一般的なエンジンが作製できる。

オブジェクトマネージャと簡単にいってしまうとそれまでだが，3Dモデリングのモーションと各種イベントに対する反応はオブジェクトにパッケージングされる。そのオブジェクトモデルをどのように規定するか，具体的な解決はどうするのかなど課題は多い。

たとえば，シンプルな3D格闘ゲーム（入力による状態の変化，相互作用）といったすでに確立されたレベルのところから動作を一般化していくとシェンムーのようになるのだろう。

オブジェクト間には物理的な力のやり取りと情報のやり取りが発生する。それぞれに対してメッセージの交換が行われ，相互に解決していくわけだ。物理的な力のやり取りを一般的に解決するエンジンが物理演算エンジンであり，情報のやり取りを一般的に解決するエンジンがAIということになる。一般解がなくとも，必要なだけのパターンを用意することで擬似的なモノは実装可能であり，現状でのゲームではそれで十分だ（それはそれでしんどいという話もあるが）。このうち，物理演算は遠からず実装されてくるだろう。簡単ではないが，解決できない問題ではない。多体問題などの解決で課題があるというのはネットワークゲームのときにも述べられていたことだが，疑似解を出すことは不可能ではない。昨今のCPUではそれらを解決するだけのパワーを備えつつある。PlayStation2のEmotionEngineなんてのもそれを見越した設計になっていると考えていいだろう。

一方，AIの解決は遠い道のりだ。推論エンジンのためのハードウェアや連想記憶のためのメモリ，データベース支援のためのCPU付加機能といったアプローチはなされていない。解決法が見えないというのが正解かもしれないが，「次の次の次世代機」あたりでは実装されるのだろうか？

とにかく，数年後にはこういった大理石の柱とわらの柱でゲーム世界は形成されることになるだろう。

見てわかるように，ゲームが最先端分野に近づいてきている。たいていはほかの研究の実地応用という感じで，最先端研究のおこぼれを吸い上げているという感じなのだが，進化の速度が違うのだ。すでに追いついてしまおうとしている。

かといって最先端の研究者と同じことをプロのゲームプログラマがやっていたら不経済この上ない。それでは採算なんか取れるはずもない。世知辛い世の中なので，そういったことを度外視できる太っ腹な企業は滅多にあるものではない。そういうのをアマチュアレベルでやろうとするのもおかしな話かもしれないが，逆にいえば，こんなのはアマチュアレベルでしかできないことかもしれないとも思うのだ。

ゲーム制作といっても，現場で要求されるスキルはどんどん変化している。スーファミ時代のスプライトからPS時代では完全にポリゴンに変わり，プログラマの世代交代が行われた。これから訪れようとしているPS2時代では，さらに違ったスキルが要求されようとしている。最近では，

「どっかに数学屋さんいませんか」

とか聞かれるようになってきている。物理モデルだのといっても，それ以前にオブジェクト指向から始めなければならない場合も多いだろう。こういう情勢になるのはとうにわかっていたことではあるのだけれど。

その次がAIの世界。一例でいえば自然会話だ。そんなものが実現できたら苦労はしないのだが，要求されている以上はクリアしなくてはならない壁だ。

まだ誰もクリアしていない分野だけに基礎研究が重要となる。本誌に連載されている知能機械概論などを読めば処理に対するアプローチが独特なものであることは理解できるだろう。

当座はなくてもなんとかなる。一定のメッセージだけ繰り返すキャラクターよりも，ある程度知識ベースを持っているほうがいいに決まっている。

「ツンさんは西のほうにいたよ」

というレベルを少しずつ洗練されたものにしていけばいい。

こういった分野で最先端（かもしれない）ところを目指している（のかもしれない）Polyphony Digitalの求人内容（Oh!X 復刊号掲載）をもう一度よく見てみよう。

アルゴリズムを考えるのが好きだ
アセンブラ・プログラマー
信号処理ならまかせろ
ゲームの企画は30以上持っている
SOFTIMAGEはばっちりだ
LightWaveなら自信あり
CG関連ツールを作ったことがある
MATHMATICAがフェイバリット
1万ライン以上のCプログラムを書いたことがある
プログラムの命はクロック数だと思う
シナリオライティングで食べていける
CADCAMを使わせたらプロレベル
タイポグラフィーは大得意
プログラムは未経験だが，高等数学はOK
物理学者になろうと思っている
NURBSモデリングには慣れている
OpenGLを使いこなせるつもり
映像演出のプロ経験がある
小説を書いたことがある
コンピュータは知らないが，デッサン力はある
アニメの原画は得意科目

……まあ，時代はまだまだ次世代機レベルで手いっぱいということか。

同人ゲームを作ろう！

福嶋 章太　Fukushima Shota

ちゃんとしたゲームを作るにはひとりでは苦しい場合も多くあります。チームで作るとなると，販売まで考えたいもの……ということで同人ゲームが浮かび上がります。ギョーカイ入りした筆者による少人数ゲーム制作のドキュメンタリ（？）＆ノウハウをお届けします。

はじめに

X680x0全盛期というと，もう6，7年前になるでしょうか？　なんてことを書くと，反対意見も多そうですが，少なくとも，筆者がX680x0をメインで使っていたのは，7年前までだったと思います。確かまだ学生をしていました。そのころの，筆者のX680x0の主な利用方法といえば，市販ゲームのプレイ，草の根BBSへのアクセス，そこで収集した画像や音楽データの鑑賞，PDS（フリーウェア）などによるマシン環境のカスタマイズといった，いわゆる普通のパソコンマニア的利用方法でした（そういえば，そのころのデータがまだMOに残ってるな……）。そして，多くのX680x0ユーザーがそうであったように，他人の作ったものを使う（鑑賞する）だけでは飽き足らずに，自分で作る楽しさに目覚めました。それが幸か不幸かはわかりませんが，筆者の運命を決定づけ，気づいたときには，ゲーム業界という泥沼の中に，どっぷり浸かっていたのでした。この記事を読んでる人のなかにも，似たような境遇の人は多いのではないでしょうかね。

X680x0は，なにかを作る環境としては，非常に優れた箱庭でした。なにを作るにしても，ある程度の道具（ソフト）と情報が揃っていました。プログラマ，グラフィッカー，サウンドデザイナーなどなど，多くのクリエイターが，X680x0という箱庭で育ち，多くの優れた作品，特に多くのゲームが生み出されました。なかには，市販ゲームにも引けを取らないような完成度の高い作品もありましたよね。

さて，昔話はこれくらいにして本題に入りましょう。この記事の本題は，何人かのチームでもってゲームを作るときの手順の紹介です。もちろんいまだにX680x0を現役で使っているのであれば，X680x0用のゲームでもかまいませんし，ビル○イツに魂を売り渡したのなら，Windows用のゲームでもかまいません（最近ようやく，Windowsでも，同人ソフトを作れるだけの道具と情報が揃ってきたように思います）。一応，読者の多くが参考にできるように，同人ゲームを作る手順となっていますが，規模が同程度であれば，市販ゲームを作るのも似たようなもんです。実際，筆者はそうやって市販ゲーム作ってますから（笑）。

どんなゲーム？

では，まずどんなゲームを作るか考えましょう。といっても，具体的なゲームの内容（ジャンルや操作方法など）を考える前に，決めなければならないことがあります。だいたい，以下のような項目になります。

1. 制作にあたるスタッフの人数
2. ゲームを実行させる環境
3. ゲームを発表する場
4. ゲームを制作する期間
5. ゲームを収録する媒体

同人ゲームであれば，基本的にメンバー全員で意見を出しあって，全員の都合がよいように決めましょう。この記事の例では，仮に以下のように決めました。

1. 6名
2. Windows9x＋DirectX
3. 同人誌（ソフト）即売会
4. 約半年
5. CD-R

さらに，使用する機材やソフトの確認，パッケージ作成にかかる費用などの確認も行っておくとよいでしょう。

では，この決定事項を基に，具体的なゲームの内容を考えましょう。これもメンバー全員で意見を出しあって決めていけばいいのですが，ただただ意見を出しあうだけでは収拾がつかないので，誰かリーダーを決めて，意見をまとめる必要があります。とりあえず，読者がリーダーになったとして，話を進めましょう。ちなみに，安易にこのリーダーを引き受けると，いつのまにか会社の社長をさせられてたりするので，注意が必要です。

とまぁ，冗談はさておき，リーダーは，各メンバーから出された意見をもとに，具体的なゲームのプランニングを行います。要するに，企画書の作成です。この記事の例では，仮にステージクリア型のパズルゲームの企画書を作ったとします。

役割分担とスケジューリング

企画書ができあがったら，次は役割分担とスケジューリングです。この段階になると，いままで率先して意見を出してくれていたメンバーも，すっかり息をひそめてしまうので，すべてはリーダーの双肩に掛かってきます（いやホント，そんなもんです）。

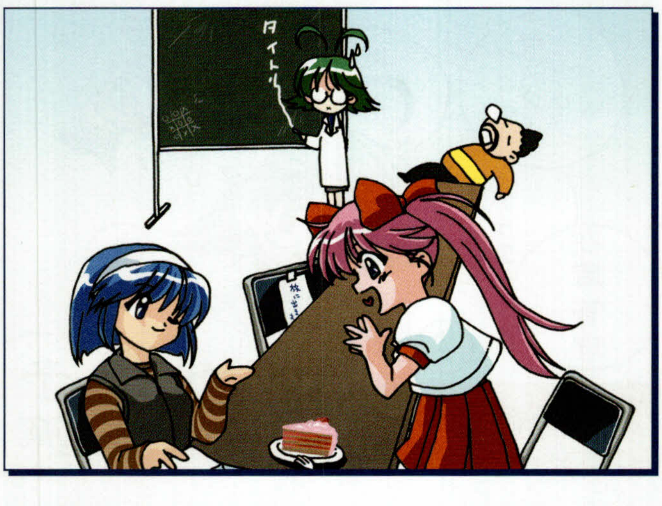
まずは，役割分担です。だいたい以下のような分担になります。同人ゲームの場合，人数が足りないことが多いので，複数の役割を兼任する人も出てくるでしょう。

・ディレクター

全体の指揮と方向性を決定する人です。普通はリーダーが担当します。

・シナリオライター

ストーリー性のあるゲームを作る場合に必要になります。お話を考え，文



章に起こす人です。

・プログラマ

企画書を基に，ゲームシステムを設計し，コーディングする人です。

・グラフィッカー

キャラクターデザイン，原画，着色，インタフェイスデザインなど，ディスプレイに映るすべてのパーツを作る人です。1名では手が足りないことが多いので，複数名でさらに細かく役割分担したりもします。

・サウンドエディタ

BGMやSEなど，スピーカーから聞こえてくる音を作る人です。必要であれば，声優の手配や録音などもします。

・データエディタ

マップの作成や，キャラクターのパラメータ設定など，ゲーム中に利用されるデータを作る人です。

・デバッガー

制作中のゲームをテストプレイする人です。同人ゲームの場合は，基本的にスタッフ全員で担当します。

役割分担が終わったら，各役割ごとにスケジューリングをしましょう。役割ごとといっても，各役割は独立しているわけではなく，それぞれが密接な係わりあいを持っていますので（絵や音がないとプログラマがプログラムを組めなかったり，システムが設計されていないと，データエディタがデータを作れないなど），作業の流れを把握したうえでスケジュールを立てる必要があります。また，役割を兼任している人がいる場合は，その人になるべく負担がかからないような工夫も必要です。サンプルスケジュールを作ってみたので参考にしてください（表1）。

さぁ作ろう！

ようやく，実際のゲーム制作にまでたどり着きました。リーダーは，ここからはディレクターとして，制作全体を指揮していくことになります。全体の進行状況を把握し，スケジュールどおりに進んでいるかどうかを，常にチェックしなければなりません。円滑に作業が進むように，細かな気配りも必要です。さらに，企画書ではカバーしきれなかった細かい仕様なども，逐次決定して制作を進行させていきます。

制作進行全体を円滑に進めるために，いちばん重要なのは，なんといっても，情報の統一です。いくら初めに企画書を作ったからといっても，作業を進めていくに従って，細かい部分の仕様はどんどん変化していきます。ディレクターは，仕様の変化を常にチェックし，実際に変更された事項をメンバー全員に，速やかに伝えなければなりません。筆者の場合は，メンバー全員が，E-Mailを利用できることから，専用のメーリングリストを用意し，そこに情報を流すようにしています。

ゲームの制作というのは，ある意味，流れ作業的な側面も持っています。例を挙げると，シナリオライターが作ったシナリオプロットをグラフィッカーが受け取りキャラクターデザインをしたり，サウンドエディタが作ったBGMをプログラマがゲームに組み込むなどの作業です。円滑に作業を進行させるには，このデータの受け渡し部分に，タイムラグを生まないような工夫も必要です。

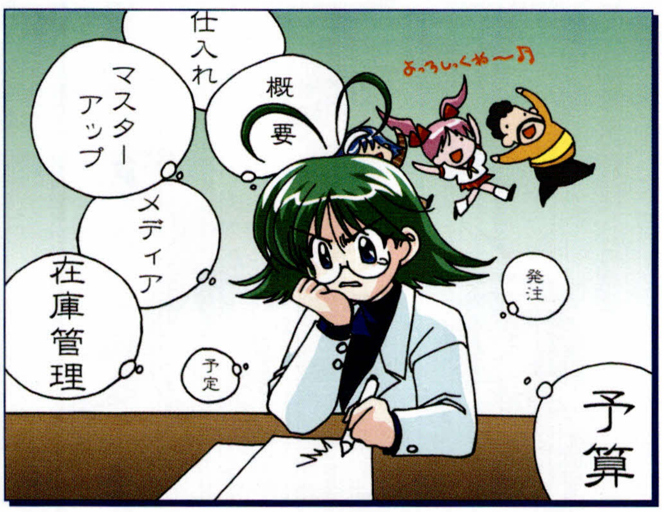
もちろん，各役割担当者同士が速やかな受け渡しを行ってくれれば問題ないのですが，筆者の経験からいって，ディレクターがきちっと管理しない限り，必ずどこかでデータの受け渡しによるタイムラグが発生します。そこで，ディレクターは，各役割担当者の進行状況を把握し，できあがり次第，そのデータを必要としている担当者への受け渡しを指示しなければなりません。

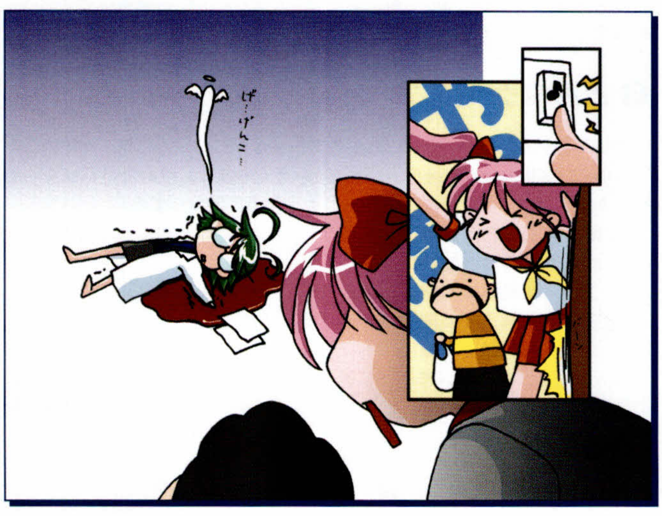
トラブル発生！

さて，スケジュールに目立った遅れもなく，順調に制作が進行してきました。このままのペースで進行させれば，予定どおりに問題なく完成させることができそうです。「よかったよかった」と，胸を撫で下ろしたいところですが，そうはいかないのが世の常です。そうです，そういうときに限ってト

	3月	4月	5月	6月	7月	8月
					CDマスターアップ→	←発売日
				マニュアル＆ジャケット→	パッケージング→	
ディレクター	企画書→					
シナリオライター	シナリオプロット→	絵コンテ→	シナリオデータ→			
プログラマ		システム設計→		α版→	β版→ マスター→	
グラフィッカー	キャラクターデザイン→		原画→ インタフェイス→	着色→		
サウンドエディター			BGM→ SE→			
データエディター			マップデータ→			
デバッガー				←デバッグ開始	デバッグ→	

表1　同人ゲームをつくろう！ サンプルスケジュール





ラブルというものは発生するのです。マーフィーさんも確か，似たようなことをいっていたので間違いありません。断言しましょう。必ずトラブルは発生します。

しかし，動揺ばかりしていても，問題は解決しません。トラブルに対して，なんらかの対処をしなくてはなりません。マスターアップ日が近ければ近いほど，迅速な対応が必要になります。では，具体的なトラブル対処法を，プロジェクト全体に対する影響が少ない順に紹介しましょう。

1. スケジュール立て直し

いちばん理想的なトラブル対処法です。スケジュールを立て直すだけで問題が解決します。しかし，これを実行するには，あらかじめ，余裕を持ったスケジューリングを行っておく必要があります。でなければ，スケジュールを立て直すといっても，無謀，もしくは実行不可能なスケジュールにしかなりません。この方法でトラブルに対処し，最終的な完成予定に間にあわせることができたディレクターは，周囲のメンバーから賞賛の眼差しを向けられることでしょう。

2. 人海戦術

トラブルが発生し，スケジュールに遅れが生じた部分に人員を投入し，作業スピードをアップさせる方法です。当然，増員するための人的資源が必要となります。幸いなことに，増員メンバーが見つかったとしても，思ったような作業スピードのアップを行うためには，相応の工夫が必要です。ひとりで行っていた作業を2人で行ったからといって，単純に効率が2倍になるわけではないので，効率のよい作業分担をしなければなりません。

3. 仕様変更

完成予定に間にあわせるために，ゲームの内容を変更します。ちょこちょこと，その場しのぎ的な変更を行っても，いずれまた変更を余儀なくされることが多いので，一度変更すると決めたのであれば，ある程度大幅な変更を行いましょう。ただし，その変更によって，スケジュールがどの程度変化するかを，しっかりと把握しなければ意味がないので，注意しましょう。

4. 最終予定（発売日など）の変更

これは，最悪のケースです。同人ゲームだからといっても，これを行うのは最終手段と考えるべきです。いちばん重症なのは，メンバーの士気の低下です。やる気がなくなります。たとえ，余裕を持ったスケジュールを立て直したとしても，作業が円滑に進行することはないでしょう。それでも，プロジェクト自体をお蔵入りにするよりはましなので，最後の最後の奥の手として用いましょう。

Column

ども，今回久々に原稿を書かせていただいた福嶋です。なんか，むちゃくちゃな文章になってしまし，お見苦しい点も多いと思いますが，なにとぞご容赦くださいませ。

えっと，本文中にも冗談で，ちらほらと出てきますが，私，小さなソフトウェア開発の会社を運営していたりします。なんで，そんなことをここに書くのかというとですね……。実は，人手が足らないんですよぉ～～（涙）。現在，Windows用ゲームを中心に開発しているのですが，春からは，"ぴーえすつー"なんて話もあって，かなり深刻な人手不足状態なんです。

でまぁ，もし，コンピュータゲームの開発に興味がある方などいらっしゃいましたら，連絡などいただけるとうれしいなぁ～なんて思っておる次第なんですよ。

いやホント，まじでお願いしますよ～～（涙）

連絡先：
http://www.quartic.co.jp/
support@quartic.co.jp

やっぱり最後は修羅場に突入

いくつかのトラブルもありましたが，なんとかβ版のプログラムも完成し，絵や音の素材もすべて揃いました。しかし，まだまだ先はあります。デバッグ，マスターアップ，量産，パッケージング，そしてそのまま即売会当日へと流れていきます。このあたりになると，たいがいどんなプロジェクトでも，締め切りが目前に迫ってしまい，寝る間を惜しんで作業をしなければならなくなります。メンバーはみな，疲労と焦りがピークに達し，眠気は吹っ飛ぶものの，妙な高揚感とともに，奇怪な言葉を口ずさむ者や，突然踊りだす者など，おおよそ仕事に使える状態ではなくなってしまいます。

しかし，リーダーであるあなたは，こんな状況にもめげてはいられません。最後の最後まで，状況を冷静に判断し，適切な指示を出していかねばなりません。もう，役割分担もなにもあったものではないので，役に立ちそうな人間を取捨選択し，手薄なパートへと投入していきます。

こうしてようやく，即売会当日を迎え，販売もなんとかこなし，即売会の終了とともに，永きにわたったプロジェクトも，終焉を迎えるのです。

あとは帰って，死んだように眠るだけです。ようやく，リーダーであるあなたにも安息の時間が訪れました。では，おやすみなさ～い。

安息は訪れた？？

あれ？　もう終わったと思ったのですが，まだ続きがあります。これはどーしたことでしょう？　発売したら終わりではないのでしょうか？　そうです。まだ終わってはいなかったのです。リーダーであるあなたには，まだやらねばならない重要な仕事が残っていました。それは，ユーザーサポートです。

どんなに時間をかけてデバッグしたとしても（たいてい時間などかけれはしないのですが），バグというものは見つかるものです。ちょっとしたスペルミスから，ゲームのハングアップバグまで，規模はさまざまですが，あなたは，可能な限りそれらのバグを駆除しなければなりません。そして，作品を買ってもらった人々からの苦情に対処しなければなりません。最近は，インターネットが普及してますので，ホームページ上で，パッチを公開するなどして対処することが多いようですが，昔なら，郵送などの手間のかかる作業をする必要がありました。同人ソフトとはいえ，お金を出して買ってもらったものですから，責任を持って，対応しましょう。

とまぁ，そんなこんなで，ようやくバグ報告のメールもこなくなり，今度こそ本当に一息つけそうです。安心しましょう。これで本当に，このプロジェクトは終わりです。お疲れさまでした……。

ん？　あ，あれ？　今度こそ本当に終わったんじゃ…？？
なにをいっているんです，もう次のプロジェクトは始まっているんですよ！
→ふりだしへもどる（涙）



VisualBasicで作る
スプライトエンジン

中野 修一　Nakano Shuichi

DirectX7でサポートされたVisualBasicからのDirectX呼び出し。ここではその恩恵に与かるためにDirectDrawによるスプライトエンジンを作成してみました。DirectDrawの使い方のサンプルとして，また手軽な高速キャラクター表示システムとしてお使いください。

喜ぶべきことがある。DirectXがVisualBasicからも使えるようになった。いや，前から使おうと思えば使えなくはなかったのだけど，正式にサポートされた。せこせこ汗を流すよりは金で解決できるものは解決する，果報は寝て待つというのが「PC」流だ。口を開けていれば牡丹餅も落ちてくる。

落胆するかもしれないことがある。全然使いやすくない。たいてい待ってれば牡丹餅が落ちてくるのだけど，おいしい牡丹餅を期待してはいけない。これも「PC」流の処世術だ。こういうのは自分でなんとかしようとするととてつもなく大変なことになる。手が出せない部分は開き直ったほうが利口である。BASICごと作ろう，とか，OSごと……という画策もないではないが，どっちにしろWindowsの流れは無視できないのだ。

それでも用意されたデモを見ると，VBでもかなりのことができそうな気になってくる。昨今のPCの性能を考えると当然なのだけど，これまではそれを引き出す方法がほとんどなかった。デモの一部はC++で書かれたものよりデキがよかったりする。

最高のものを求めるならBASICなんて使ってはいけない。でもちょちょっといじってそこそこのものができるとなれば，いろいろ遊べる環境ができる。ソフトがタコで多少遅くても，あとはハードパワーでゴーゴーってのが「PC」流だ。よし，まったく問題はない。

今回は付録CD-ROMにDirectX7.0a SDKが収録されている。フルインストールするとかなりのHD容量を食うのだけど，とりあえずはフルインストールをおすすめする。デモを眺めて，ヘルプでも見て，飽きていらなくなったらアンインストールすればいい。とにかく，このSDKによってVisualBasicのDirectX制御が可能になるのだ。とりあえず，入れるべし。

さて，あまり知られていないことかもしれないが，このDirectX対応はVB5CCEでも使用できる。念のために書いておくと，VB5CCE（VisualBasic 5 Control Creation Edition）はマイクロソフトがWeb上で配布している，ActiveXコントロール作成用のVisualBasic開発環境だ。単独で動作する実行ファイルは出力できないものの，HTMLファイル上に張り付ければ動作するActiveXユーザーコントロールを制作できる。コントロールとはいっても，用途に制限はないのでプログラムさえ作ってしまえばなんでもできる。実行ファイル形式での開発もできなくはないので（最終的なファイルはできないが），デバッグ終了段階までVB5CCEで作っておいて製品版のVisualBasicを持っている友人に「ディストリビューションパッケージ作っといて」とプロジェクトごと渡せばそのまま実行ファイルを得ることもできるだろう。VB5CCEは以下のURLからダウンロードが可能だ。DirectXを使うにはVB5CCEインストール後にDirectX7 SDKをインストールすればよい。

http://www.microsoft.com/japan/developer/vbasic/controls/controls.htm

とにかく，VB5CCEでもDirectXの開発ができるというのは，非常にうれしいことだ。CD-ROMに入っているサンプルは最終的にVB6でコンパイルしたものも入っているけれど，私は通常はBV5CCEしか使っていない。普通のVBだとでかいのでノートパソコンなどには入れたくないけど，VB5CCEならとりあえず入れておこうかという気にもなる。

Column

●かなり余談

VB5CCEで使用するモジュールが増えていくと，なんとなく動作が不安定になっていくよーな気がする。エディット後に実行すると100％飛ぶ環境のマシンもあったり，いろいろ苦労している。使用環境にもよるのだろうが，3台分，うちひとつはかなりクリーンなマシンなのだが，それでもかなり似た症状が出る。

これでは開発にならないのだが，慣れというのは恐ろしいもので，だましだまし使ってればなんとかならなくもない。うちでは，プログラムを一度実行して，一時停止状態でエディットして再実行などを行うと安定して動作するようだ。ただしセーブし忘れることが多いので注意。以上，他人にはまったく役に立たないかもしれないTIPSだ。

●スプライトエンジンを作ろう

「VBでDirectXが使える」というのでDirectXもずいぶん身近になったかというと，前述のとおり，なかなか手軽とはいかない。

幸い，DirectX7 SDKにはサンプルがいくつか用意されているので，それぞれのDirectXファンクションごとに使い方を見ていくといいだろう。ちょこちょこと書き換えて組み合わせるというのが基本だろうか。サンプルの記述はちょっとわかりにくい点も多く，なにより必要以上にややこしそうに見える。ここではお決まりの処理をまとめてもっと手軽に扱えるようなシステムを作っていってみたい。

「DirectXサポート」は，サンプルプログラムを見ればわかるように，DirectXのAPIがそのまま関数に落ちてるだけみたいな感じで，BASICレベルから使うには，あまりうれしい状況ではない。DirectXの基礎から理解していないと手を出せない状況だ。問題点は2つ。第1に多機能すぎるという点だ。機能は多いほうがいいという考え方もある。しかし，さんざんあちこちを初期化しまくった挙句に「Hello World」とか出力するようなシステムはあまりに馬鹿らしい。第2に，これはVisualBasicだということだ。アプリ部分を書くのはともかく，システム寄りの部分を記述するのは適当ではない。そういうのはやはりちゃんと用意しておいてほしいものだ。

もうちょっと高位な関数とかをつけられなかったのかという気もするが，下手に変な高位なものを用意されるとまたそれはそれで困りそうなので，基

本部分だけ揃えたのは正解なのだろうか……。

ということで，基礎部分は用意されたのだから，あとは上位階層をつけ加えてやればいい。で，なにが面倒かというと使うまでの手間が多いということだ。いや，DirectX7相当だから，これでもかなりシンプルな記述になっているそうなのだが……。その辺をシンプルにして，ゲームで使いそうな機能を揃えていくことにしよう。

さて，いまやゲームというとDirect3D関係が全盛だけど，やはり基本は2D。DirectDrawが手軽に使えるようになればそれなりにおいしい状況には違いないのだ。ゲームというわけではなくて，デスクトップアクセサリのようなものも作りたい……ということで，まずは2D環境の整備から始めよう。2Dといえばスプライトがなくては始まらない。ここではDirectDrawでスプライトっぽいものを使えるようにするための補助関数群を作っていくことにする。スプライトエンジン自体をVBで記述するのでパフォーマンスをあてにしてはいけない。

本当はコントロール化してさらにスマートにする予定だったのだが，コントロール上での表示実験に失敗してちょっとハマってしまっていることと，間接呼び出しになるのでパフォーマンスの問題が懸念されるなどの点を考え合わせて，標準モジュールによるスプライトエンジンとゲームのスケルトンという形で提供することにした。綺麗なブラックボックス化ではないけど，メインルーチン部分のモジュールを書けばいろんなゲームに対応できるほか，スプライトエンジン自体も細かい部分をカスタマイズしやすいので，これはこれで正解かなと思っている。

●スプライトエンジンの仕様を決める

DirectDrawでできることとはなんだろう？

DirectXでいう「スプライト」というのは「マスク付きパターンを描画できます」というのと同義なので，非常に低レベルのものといわざるをえない。それくらいのことはWindows GDIでもできるのだけど，パターンの抜きをやろうとすると，マスクパターンを用意したうえで3回も書き込みしなくてはいけない。とはいえ，ビデオチップというのは昔からそういうのに最適化されているハードウェアなので「DirectDrawだから速い」というものでもない。マスクの処理などを除けば，昔から2D性能は飽和したといわれていたとおりだ(それでも年々確実に速くなってるのも凄いが)。

Windowsのグラフィック APIは初めて見ると信じ難いほど四角い箱しか描けないようにできている。箱を組み合わせて，そこに文字を書くという程度。あとは線を引いたり円を描いたりといったプリミティブな描画コマンドだけだ。複雑な形やグラフィックパターンを描くということに関しては恐ろしく弱かった。使えそうなAPIを探して試すと，驚くほど使えない仕様(16×16ドット限定とか，単色描画のみとか)になっている。DirectDrawはその辺を強化したAPIだと思えばいい。逆にいえばグラフィックパターンの描画しかできない。なーに考えてるんだぁ的なシステムが実にWindowsスタイルである。ビデオチップはWindows GDIの処理を行えるので円弧発生などのハードウェアを持っていたとしても，DirectDrawサーフェス上に描画することはできない。GDIとDirectDrawは概念的には相互を補完する関係にあるが，組み合わせて使うことはできない，もしくはやっても非常に非効率的なのだ。

さて，DirectDrawの基本を把握しておこう。DirectDrawではサーフェスとして定義されたメモリ領域に対してグラフィックBitBLTの描画処理を実行する。ちなみにBitBLTは「ビットブリット」と読む。平たくいえば，グラフィックパターンの転送処理のことだ。

サーフェスの大きさはそれぞれ任意に確保できる。この領域はVRAM上でもかまわないし，メインメモリ上(システムメモリという)でもかまわない。いくつ作ってもかまわない。画面に表示されるVRAM部分もサーフェスで，バックバッファもサーフェスだ(これらはさすがに必ずVRAM上に取らないといけない)。この2つの特殊なサーフェス以外に，ユーザーサーフェスを作っておき，そこに描いてあるグラフィックパターンを，特定の色を透明色と見なしてほかのサーフェスに転送合成することができる。サーフェス内部の任意の矩形領域を，ほかのサーフェスの任意の矩形領域に転送可能だ。表示領域と同じ大きさのバックバッファを用意しておき，そこにパターンを次々に描き込んで，全部描き終わったら表画面と入れ換える。

DirectDrawの基本はこれだけだ。アーケードゲームなどで使われているフレームバッファ形式のスプライトシステムだと思えばいいだろう。つまり，グラフィック処理と同程度の自由度を持って仕様が決められるということだ。

スプライトエンジンの仕様は「X68000くらい動けばとりあえずよし，できればもっといろいろつけ加える」という方針で決めたい。世の中X68000ユーザーばかりではないのだろうけど，まともにスプライトの扱えるマシンがほかになかなかないし，どれくらいのことができるかもよくわかっているので仕様も決めやすいのだ。

●管理スプライト数

暫定的に256個としておいた。

何個でもいいんだけど，X68000が128個のスプライトだったのでこれくらいあれば当座は困らないのではないかと。これでは志が低いと思われるかもしれないが，それぞれのスプライトの大きさは無制限で，ひとつのスプライト中にアニメーションパターンを含むことができる。スト2タイプのゲームでいうと，キャラクター1，2，それぞれの飛び道具，背景や特殊効果，その他で10数～20個程度のスプライトでできる感じだと思えばいい(効率は無視。たぶん，いちばんスプライトを食うのはスコアカウントの数字だろう)。

ちなみにエンジン部の数字をちょこっと書き換えればいくらでも増やすことは可能なので，どうしてもスプライトで囲碁を作りたいという人は勝手にやってほしい(後述のとおり，現仕様のままでも表示数は無制限だけど)。

表示数が無制限であるにも関わらず，スプライト個数に制限があるのはなぜかといぶかしむ人もいるかもしれない。これはシステムで管理するスプライトとそうでないものの差だ。全部グラフィックパターンとして自前で管理する人には画像転送だけでいいのだろうが，それだけでスプライトと呼ぶのはちょっと抵抗がある。スプライトエンジンに管理された256個分のパターンについては明示的に更新処理を指定しなくても自動的に表示が行われるようになっている。エンジンの管理外のパターン描画については自由という意味だ。表示位置の管理や書き換えは各自で処理してもらうことになる。

●拡大/縮小

縦横独立の倍率指定というか，表示サイズの指定が可能だ。ドット数で指定する。疑似3Dスクロールとかの応用はすぐに思いつくだろうが，特殊効果的に使うのが正解ではないかと思っている。ブロック崩しにしても，ブロックに当たった瞬間ボールがぽよよ～んと弾むとかなり質感が変わってくるのだ。餓狼伝説っぽい処理もやろうと思えばできなくはないだろうけど管理は面倒だ。

●BG(背景)

スプライトシステムだからBGがないと始まらない。今回は簡易版ということで，2枚の背景(それぞれ1枚絵)のトーラススクロール対応版とした。

●直接描画

スプライトの管理外で直接サーフェスにBLTできるようにしたものを用

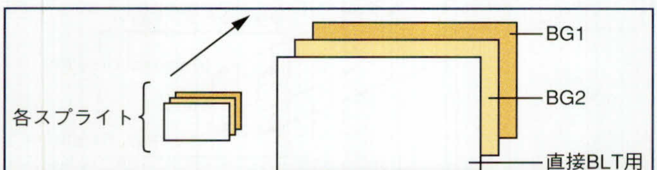


図1　今回のスプライトエンジンのプレーン構造

意した。具体的には，背景プレーンをもう1枚取り，そのプレーンに好きなだけ書き込んで合成する方式となる。無数に描画を行いたいときなどに使う。

今回見送ったもの

●チップBG

チップパターンでBGを構成するもの。X68000でいうところのBGのようにスプライトパターンを並べて背景を作るタイプ。こっちのほうが使いではあるのだけど，このスプライトエンジンではBGは単に1枚絵を2枚置いているだけだ。スクロールタイプの2DRPGなどは不可能ではないが，単純にやると凄くメモリが必要となる。さすがに手を抜きすぎたかなという気もしている。

●縦横反転，回転など

DirectDrawのCapabilityを見ると，ビデオカードごとにサポート状況がまちまちだ。それでもどれもHELで見るとできそうなのだが，なぜかうまくいかなかったので割愛。

●特殊効果

DirectX7になってもAlphaつきのBLTはサポートされなかったので，当初記述していたBLTFX用の設定はBLT用に書き直した。仕様上はいろいろな効果が実現できるはずだけど，ドライバのサポート状況はなかなかお寒い限りで，TNT2やGeFORCEでさえ未サポートファンクションが多い。結局のところ，DirectDrawのサポートは各社とも熱心ではなく，Direct3Dを使ったほうがよい結果が出そうだと感じた。

●プライオリティ

BG間，BGとスプライト間，などのプライオリティが現在設定できない。BG1がいちばん奥，ついでBG2，スプライトは番号の若いものほど手前に表示される。このあたりは拡張する予定だったのだが，すっかり抜け落ちてしまった。バケツソートの話は入れたかったのだけど……。次期バージョンではスプライトごとに256段階で自由に優先順位が設定できるようにする予定。

DirectDraw使用の流れ

唐突だが，DX7 SDKのサンプルはわかりにくいと思う。サンプルによってオブジェクト名とかバラバラだわ，プログラム中で唐突に変数宣言してるわ，あまりよい例題ではないと思う。自分のプログラムがわかりやすいほうではないというのは重々承知しているが，これはちょっといただけない。ざっと流れを確認しておこう。

●オブジェクトの初期化

ということで，DirectDraw使用時の全体の流れを追ってみたい。まずはオブジェクトの宣言だ。

- DirectX7オブジェクト
  使いますよということで呼び出す。
- DirectDrawオブジェクト
  同上。
- Descriptionオブジェクト
  サーフェスを作るための定義部分。使い捨て可。
- Surfaceオブジェクト
  画像本体部分。
- Clipperオブジェクト
  ウィンドウのクリッピングをしてくれる。

こういったものを決まった手順で呼び出すようにする。たいていはお決まりのもので大丈夫だ。サンプルプログラムからそのまま抜いてきていじろう。ウィンドウハンドルとか出てくるとちょっと引いてしまう人も多いかもしれないが，凝ったことをしなければあまり気にしなくてもいいだろう。

●サーフェスの確保

オブジェクトの初期化が終わったら，サーフェスを確保する。Descriptionオブジェクトの各プロパティを設定し，実際に領域を確保する。

```
ddsdBLT.lFlags = DDSD_CAPS Or DDSD_WIDTH Or
DDSD_HEIGHT
ddsdBLT.ddsCaps.lCaps = DDSCAPS_OFFSCREENPLAIN
ddsdBLT.lWidth = scXsize
ddsdBLT.lHeight = scYsize
Set objDDBLTSurf = objDD.CreateSurface(ddsdBLT)
```

のような感じの指定となる。なお，今回はファイルからの画像を直接指定するタイプのものを多用している。

●BLT

こうして作ったサーフェスを，画面イメージとなるサーフェスに書き込んでいく。この場合透明色を指定してマスク処理を行うこともできる。今回のエンジンではマスク色は黒（RGB = #000000）に固定されている。

BLTFastとBLTが使用されているが，BLTFastはパラメータが簡略化されたBLTだ。逆にさらに多機能なBLTFXもある。

```
BLTFast　ソースRECTと書き込み座標を指定
BLT　　　ソースRECTとデスティネーションRECTを指定
BLTFX　　BLTの指定に特殊効果フラグを追加
```

要は，BLTFastだと元画像のまま，BLTだと任意の大きさに拡大/縮小できる。FXまでつくと反転とか回転とかいろいろできるようになる（ただしドライバ側でサポートされている保証はない）。

ハードサポートされていれば特に速度は変わらない。ソフトエミュレート時はBLTFastがもっとも高速だ。

ここで使われるRECT型という構造体はDirectXサポートで多用されるもので，ウィンドウ上の矩形の，Top，Bottom，Left，Right情報をまとめたものだ。つまり（x0，y0）－（x1，y1）の矩形がRect1で定義されているなら，

```
Rect1.Top = y0
Rect1.Bottom = y1
Rect1.Left = x0
Rect1.Right = x1
```

のようにアクセスされる。

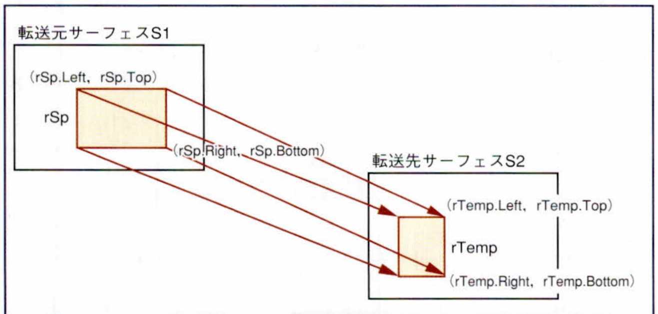


図2　S2.Blt rTemp,S1,rSp，DDBLT_WAITの動作

●フリップ

要するに，裏画面の内容を表画面に回す作業だ。ウィンドウモードでは概念的に正規のフリップ処理とは異なり，BLTFastで裏画面を丸ごとプライマリサーフェスに転送することになる。

以降はBLTとフリップを繰り返していけばよい。

スプライトエンジンの使い方

では今回作ったエンジンのだいたいの使い方を解説していこう。今回用意したサンプル（gametest02）を開くと3つのコード部分が見えるが，これらは，

ウィンドウ処理など
スプライトエンジン本体
ゲーム記述部分

に分かれる。ユーザーはゲーム記述部分だけを書けばよい。サンプルをちょこちょこいじっていくのが正解だ。初期化の順番などが入り組んでいるので，全体の流れは自ずと決まってくる。

●スプライトとはなにか

ここでいうスプライトでは，ひとつのスプライトが1キャラクターに対応する。画面上で動くキャラクターをそれぞれスプライトと呼ぶ。これらは0～255の番号で管理され，

1番のスプライトを100，150に移動する

といった感じで使われることになる。

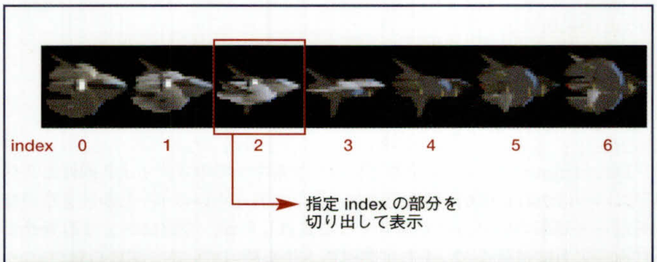


図3 スプライトパターンファイルの例と表示法



図4 パターンを横につないでスプライトファイルを作る

グラフィックパターンとスプライトは分けて管理される。これだと同じパターンで違うキャラを作ったりする場合に効率がよくなる。要するに，同じ形状のザコキャラ3つが現れるときに，同じグラフィックを3つ持つのは効率悪いね，という話だ。

まず，パターンを定義しておき，スプライトを作成するときにそのパターンから画像を選ぶようにすることになる。

それぞれのパターンは，ひとつのパターンを横に複数個並べた構造になっており，アニメーションなどを統一的に扱うことが可能だ。内部の何番目のパターンを使うかは.indexプロパティで指定する。

なお，今回使ったのはただのBMPファイルだが，256色ランレングス圧縮BMPを使ったので，ファイルサイズもさほどひどい状況ではない。お約束で拡張子だけSPRに変更されている。それだけでもそれなりの雰囲気になるから不思議だ（読み込みには支障ない）。もちろん，24ビットベタBMPでも問題ない。なお，この部分ではGIFなどは指定できないようだ。PictureBoxではGIFやJPEGも画像としては読めるのだが，こういうところには使えない。BMPと完全に同等に扱えるようになったわけではないらしい。

それぞれのスプライトはXSP()という配列で示され，スプライト用の構造体を持つ。これは，

```
Public      Type        XSPDAT
Pat         As Integer  パターン番号
Active      As Boolean  使用フラグ
Index       As Integer  アニメーション番号
MaxIndex    As Integer  最大アニメーションパターン数
destX       As Long     表示先X座標
destY       As Long     表示先Y座標
 Xsize      As Integer  表示時のXサイズ
Ysize       As Integer  表示時のYサイズ
Hmirror     As Boolean  水平反転
Vmirror     As Boolean  垂直反転
srcRect     As RECT     元画像のレクタングル
hitRect     As RECT     衝突判定用レクタングル
Zindex      As Integer  プライオリティ設定用
End Type
```

で定義されたもので，

```
XSP(1).distX
```

のようにそれぞれのスプライト別にアクセスすることになる。HmirrorやZindexなどは将来の拡張用だ。

・sp_init命令
便宜上必要。スプライトを使用する宣言ということで。

・sp_def命令
スプライトパターンを定義する。
sp_def パターン番号，ファイル名，キャラの横ドット数
で指定する。縦ドット数は自動的に登録される。

・sp_set命令
sp_set スプライト番号，パターン番号，横幅，縦幅，当たり判定エリア
で指定する。

・sp_move命令
スプライトを指定位置に表示する。
sp_move X座標，Y座標，スプライト番号
という指定となる。スプライト番号が先のほうがよかったかもしれない。消したいときは画面外へ。

クリッピング

Clipper Objectはウィンドウ関連のクリッピングを行うものだ。しかし実用面でいうとDirectDraw自体の表示ではクリッピングは行われない。どうしているかというと，なかなかお利口さんで，ウィンドウ外にはみ出しそうなものは丸ごと非表示にしてくれる。DirectDrawの名誉のためにいっておくと，ちゃんと設定されたエリアでクリッピングは処理されるのだ。しかし表示エリアのエッジにかかった途端に全体がぱっと消える感じだ。エリア設定でのクリッピング処理を書いた人の意思はウィンドウとのクリッピングを書いた人には伝わらなかったらしい。通常，クリッピングエリアはウィンドウ枠で取ることが多いので，実用上はまったく使えない。ウィンドウの真ん中に小さく表示するときなどはクリッピングが利いているのだけど，今回はスプライトの大きさが無制限なので，マージンを取っての処理（それもダサイが）もできない。しかたがないので，BASICでクリッピングを行う。はみ出さない部分までの画像を切り出してクリッピング面にあわせて張り付けてやる。BASICにやらせることじゃないな，普通。

考え方は単純だ。はみ出たところを切り取ってやればいい。あとは理屈どおりに記述するだけだ。

BG画面の制御

BG画面は1枚のBMPファイルで作られる。大きさに制限はないが，あまり大きいとVRAMやシステムメモリを圧迫するので注意が必要だ。

背景の大きさは元ファイルの大きさではなくて，読み込み時に大きさを指定するようになっている。たとえば，100×100ドットの元パターンを500×2048ドットに拡大して背景にするといったことが可能だ。

BG 2面はそれぞれ独立してトーラス（球面）スクロールが可能になっている。要するに「上下左右はつながっていますよ」ということだ。この辺の処理のため，BG 1枚につき最大4回描画が行われることになる。処理部は若干込み入っているが，使うだけならスクロール座標を渡してやるだけでいい。BG2については透明色（黒）を用いたマスク処理が可能だ。

・**MaxBG　システム変数**
使用するBGの最大枚数を設定する。1または2を指定。

リスト1　BLTとクリッピング

```
Sub BltSP(ppx As Long, ppy As Long, ppsp As Integer)
    'blt to the backbuffer from our  sprite
    'use the color key on the source - (our sprite)
    'wait for the blt to finish before moving one
    Dim X, Y, lx, ly, rx, ry, dx, dy As Integer

    dy = XSP(ppsp).Ysize
    dx = XSP(ppsp).Xsize
    rSp = XSP(ppsp).srcRect
    rSp.Left = XSP(ppsp).Index * SPpat(XSP(ppsp).Pat).Width
    rSp.Right = rSp.Left + SPpat(XSP(ppsp).Pat).Width
    X = ppx
    Y = ppy

    lx = X: ly = Y
    rx = X + dx: ry = Y + dy
    If X < 0 Then X = 0: lx = 0: rSp.Left = rSp.Left - ppx
    If Y < 0 Then Y = 0: ly = 0: rSp.Top = rSp.Top - ppy
    If X > scXsize - dx Then rx = scXsize: rSp.Right = rSp.Right - ppx - dx +
    scXsize
    If Y > scYsize - dy Then ry = scYsize: rSp.Bottom = rSp.Bottom - ppy - dy +
    scYsize

    rTemp.Left = lx
    rTemp.Top = ly
    rTemp.Right = rx
    rTemp.Bottom = ry
    objDDBLTSurf.Blt rTemp, SPpat(XSP(ppsp).Pat).Surface, rSp, DDBLT_KEYSRC Or
    DDBLT_WAIT
End Sub
```

・**BG1.filename/BG2.filenameプロパティ**
BGに指定したBMPデータを読み込む。

・**BG1.Width/BG2.Widthプロパティ**
BGの横幅を設定する。

・**BG1.Height/BG2.Heightプロパティ**
BGの縦幅を設定する。

・**InitBGSurfacesコマンド**
設定した内容でBGを確保する。お約束ということで。

・**BG1x/BG2x/BG1y/BG2yシステム変数**
BGそれぞれの表示先頭座標を指定する内部変数。

・**GBhomeコマンド**
BG1x/BG2x/BG1y/BG2yの値に従って実際にスクロールを実行する。明示的に座標を書き換えないと前回の座標が残っているので注意（たいていは残ってるほうが便利だけど）。

パターン書き込み

ユーザーが直接パターンをBLTするために用意した。わかっている人ならすべて自前でやってもいいだろうが，スプライトを別途書き込むとしたほうがわかりやすい人も多いだろう。スプライト番号単位での指定なので，スプライトの制御（indexによるアニメーションなど）はそのまま適用できる。

これらは独立したサーフェスに書き込まれる。スプライトエンジンの管理外であり，明示的にサーフェスをクリアしないと消えないので注意。パターンを好きなだけ書き込んでも大丈夫。もちろん，透過色処理によるマスク合成も可能だ。このプレーン以外のBGやスプライトはこのプレーンの内容に関係なく管理されているので，クリアや書き直しなどについての心配は無用だ。書きっぱなしのプレーンなのでグラフィックツールとかもできなくはない（αつきBLTがちゃんとサポートされてればなあ……）。

・**BLTSP命令**
スプライト番号を指定して直接描画を行う。

キー入力

DirectInputのサンプルを参考にカーソルキーでのスティック操作とスペースバーほかのトリガー処理を追加した。これくらいのキーしか読まないならもっと効率のいい方法がありそうな気はしたが，今回はあまり追究せずにキー入力構造体をゲットして調べる方法を使っている。関数自体はstick()という名前だ。DIStateを取るような方法を使っている範囲ではそれなりに効率のよい判定になっているとは思う。select文やif文の入れ子にしていないのは同時押しに対応するためだ。どう押しても辻褄はあうようになっている。

関数の戻り値としてはテンキーに相当するスティック操作が設定されている。同時にトリガー判定もやっているのだが，この結果はPublicな変数

リスト2　stick()関数

```
Public Function Stick() As Integer
    Static k As Integer
    DIDEV.GetDeviceStateKeyboard DIState
    If DIState.key(208) <> 0 Then k = 2 Else k = 5  'Down 2
    If DIState.key(203) <> 0 Then k = k - 1         'left 4
    If DIState.key(205) <> 0 Then k = k + 1         'right 5
    If DIState.key(200) <> 0 Then k = k + 3         'up 8
    Stick = k
    If DIState.key(57) <> 0 Then DItrig0 = True Else DItrig0 = False   'Space
    If DIState.key(42) <> 0 Then DItrig1 = True Else DItrig1 = False
      'shift(right)
    If DIState.key(44) <> 0 Then DItrig2 = True Else DItrig2 = False   'Z

End Function
```

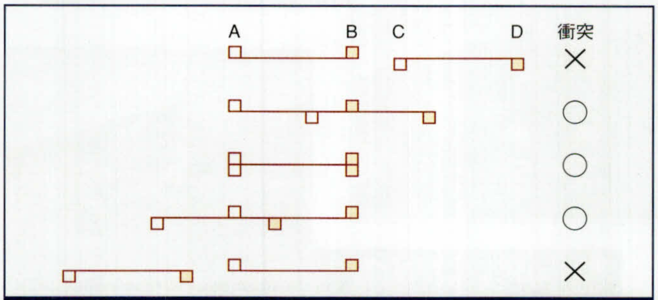


図５　線分ABとCDの取りうる状態を考える

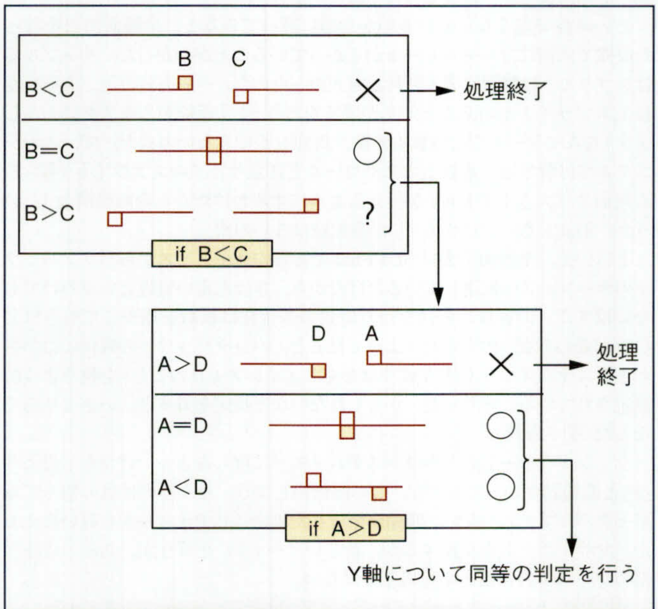


図６　当たり判定の例。X軸から一次元上で各エッジを比較する

に叩き込まれている。別の関数で処理するべきかもしれないのだが，判定のたびにキー入力構造体を取り直すのも（真っ正直に処理されると思う）どうかと思うし，keyget () などという関数を作って先に構造体を取り込んでおいてそれぞれを処理するという手だと，単なるお約束でkeyget () なりの関数を呼ばなくてはならないのが面倒だし，たいていはスティックとトリガーは同時に参照されるということでこのような仕様にした。あまり綺麗な構造ではない。トリガーだけほしいときにもスティックが判定され，スティックだけでいいときもトリガーがついてくる。そういう場合にはちょっと無駄だけど，レアケースだということで。

なお，こういう仕様だと理解してないとちゃんと使えないので十分納得しておいてほしい。

・stick () 関数

返り値１～９のテンキーに相当するスティック値。無入力は５としている。

・Dltrig0～2システム変数

boolean型で押されているとTrue，押されてないとFalseを返す。トリガーは，

トリガー０　スペースバー
トリガー１　右シフトキー
トリガー２　Z

に設定されている。stick () 関数のあとに読み出すこと。

リスト３　当たり判定

```
Function HitCheck(Ra As RECT, Rb As RECT) As Boolean
If Ra.Left < Rb.Left Then
    If Ra.Right > Rb.Left Then
        If Ra.Top < Rb.Top Then
            If Ra.Bottom > Rb.Top Then HitCheck = True Else HitCheck = False
        Else
            If Rb.Bottom > Ra.Top Then HitCheck = True Else HitCheck = False
        End If

    Else
        HitCheck = False

    End If

Else
    If Rb.Right > Ra.Left Then
        If Ra.Top < Rb.Top Then
            If Ra.Bottom > Rb.Top Then HitCheck = True Else HitCheck = False
        Else
            If Rb.Bottom > Ra.Top Then HitCheck = True Else HitCheck = False
        End If

    Else
        HitCheck = False

    End If

End If

End Function
```

当たり判定

必須のものではないと思うが，一応，当たり判定関数を用意しておいた。スプライト番号同士での相互衝突判定と２つの矩形間（RECT型を２つ渡す）の２種類だ。なお，スプライトはそれぞれ当たり判定エリアを設定するようになっている。面倒ならスプライトと同じRECTを使っていいと思うが，パターンの内部で設定したほうが普通は自然な感じになる。基本は敵キャラの当たり判定はデカく，自機の判定は小さくだ。

当たり判定は矩形と矩形の交差判定で行っている。やや面倒かなとも思ったが，必要な条件判断は４回でしかない。思いついた２通りの判定法のうち，わかりやすそうなほうを採用している（より効率よさそうなのは図示しておく）。こういうのは細かくやっても効率悪いのでif文で一気に処理している。最短２回の条件判断で終了，最悪でも４回だ。さらに縦シューと横シューではXとYの判定順を変えるなど，多少のチューニングは可能だ（接触しない確率の問題）。ウィンドウのプロポーションから自動判定できなくもないが，条件判断が１回増えるのでエンジンを直接書き換えたほうがいいだろう。

これらはあくまでも矩形単位の判定にすぎない。Windowsのような時間管理のきちんとできないシステムでは，すり抜け防止などで処理内容に応じたルーチンが要求されてくるだろう。そういうときは自前で当たり判定ルーチンを用意してほしいのだが，これらはその場合のサブ関数として使えなくもないだろう。

・HitCheck () 関数

RECT型変数２つを渡すと衝突判定して，boolean型で値を返す。ただし，現状ではx0，x1とか変なパラメータが交じっているので使う前にエンジン部を書き直すこと。

・HitCheckSP () 関数

スプライト番号２つを指定して渡すと衝突判定して，boolean型で結果を返す。

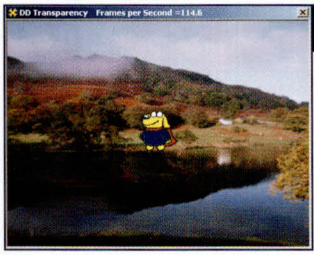


図7-a SDKサンプルの例。タイトルバーに注目



図7-b キャラクターを増やすと途端に重くなる

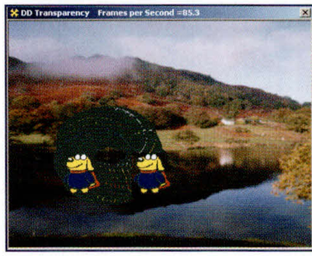


図7-c クリアをなくすと軽い



図7-b バックバッファをVRAMから取ると凄く速い

●サウンド周り

サウンドをバッファに登録しておいて，バッファを再生する。バッファは適当にいくつでも確保して鳴らせばいいだけだ。DirectXってえものの基本的な思想が垣間見える。サウンドは16種類（17だけど）扱えるように初期設定しているが，配列定義の添え字を必要なだけ増やせばそのまま動作するはずだ。

今回のBGMは手抜きで1曲をただ流すだけ。BGMの切り換え処理などをしたい局面もあるだろうが，そういう人はたいして難しい話ではないので別途対処してほしい。所詮，DirectMusicではZ-MUSICほどの柔軟な処理は望めないし，効率も悪い。まあCPUパワーでカバーってとこだ。ゲームのBGMについての考察は甚だ甘い。

ちなみに，うちの環境だとサウンドなしで動作させる分にはメモリリークもなく，長時間でも安定して動作しているのだが，サウンドを入れるとひと晩くらいでシステムごと逝ってしまうことが多いようだ。

・LoadSound 命令

効果音を読み込む。

LoadSound バッファ番号，ファイル名

で呼び出し，DirectSoundのバッファにWAVファイルを読み込む。

・DSound（バッファ番号）.Play

バッファ内の音声を再生する。プログラムではその後ろのパラメータに，

DSound(1).Play DSBPLAY_DEFAULT

などと難しいことを書いているが，要はループ再生するかどうかのフラグを書くだけだ。つまり単再生なら，

DSound(1).Play 0

でいい。

・LoadMIDI 命令

BGM要MIDIファイルを読み込む。直後にパラメータでMIDIファイル名を指定する。

・PlayMusic 命令

LoadMIDIで読み込んだファイルを演奏する。

●高速化？

開発も大詰めになったところで問題が発生した。なんのことはない。とにかく遅いのだ。動作テスト用に用意したマシンで試すと，SDKのサンプルのままだと110fps程度は出ているのだが，10fps以下という惨憺たる結果だ。ちなみにCD-ROMに収録されているgametest01は修正前のものだ。

これだと開発に使っていたノートパソコンのほうが速い。PentiumIII/450MHz＋TNT2のサブマシンではそこそこ出てたので，こんなもんかなと思っていたのだが，メインマシンをVoodoo3 3500TVにしたときに超絶的な遅さを見せられ，調べると画面解像度や背景枚数などでかなり変わってくるのでVRAM容量が少ないため（Voodoo3はフレームバッファが8MBしかない）と結論づけていたのだが，GeFORCE256に換装してもひどい結果しか出ないことにしばし愕然。この手の症状ではありがちなことだが，グラフィックアクセラレータのパフォーマンス設定を3つくらい下げる（DirectXアクセラレーションを無効にする）とそれなりの結果が出てくる。20～30fpsならゲームはできなくはない。

どの時点で遅くなっているのかを順に遡ってみると，全画面消去を行った段階で大幅にフレームレートが下がっていることがわかった。サンプルではスプライトは個別に消去される方式だったのだ。その方式だと，ある程度以上スプライトが増えると効率が悪くなるうえ，「背景が2重スクロールします」なんてゲームだと無駄も無駄，馬鹿しかやらない方式だ。で，このシステムでは背景は2重以上のスクロールと任意サイズのスプライト（もちろん画面より大きくても可）が扱えるようにするわけだから全画面消去（というか，消去しないだけかも）しか選択肢はないのだ。

とはいえ，全画面消去はBLTFastで背景のサーフェスからバックバッファのサーフェスへ転送しているだけだから，さほど重い処理というわけでもないはずだ。同容量の転送を行う背景の多重化は枚数を増やしてもさほど負荷が変わらないのも変だ。よーく見ると，バックバッファの確保にはわざわざシステムメモリ（VRAMではなくてメインメモリのこと）を使うように設定されている。そりゃ遅いかもしれないなと指定を外すと，いきなり速くなった（図7参照）。

「ウィンドウモードのときはシステムメモリに取らなきゃいけなかったような」と菊地君がいってるけど，この際無視しよう。そんなのは私の知ってるバックバッファじゃない。問題が発生した場合はビデオカードを買い換えるか，マザーボードを交換するか，新しいノートPCを買うか，あるいは該当部のソースを書き換えて対処してほしい。

●とりあえず使ってみる

リスト4はCD-ROMのサンプルのゲームコード部を簡略化したものだ。ほぼひととおりの機能が使用されている。ゲームではないし，自機も操作できないが，キー入力のようすや当たり判定，効果音も確認できる。

初期化など，全体的な構成はこれをスケルトンとして組み立てていけば大きな失敗はないはずだ。最初にRECT型で当たり判定エリアを定義している。今回のキャラクターは32×32ドットの大きさのものを使用しており，それの端から5ドットずつ内側に入ったエリアを当たり判定エリアとして登録し，いろんなスプライトで使い回している。実際にはもっとキャラクターの形に沿ったレクタングルを定義して使い分けることが望ましい。



図8 当たり判定の例。X軸から一次元上で比較する

なお，今回のキャラクターは以前，菊地君のJava版ゲームで使用されたもので，DoGA L2でモデリング，レンダリングされたものだ（L2の機能は使ってないのでL1でも同等のものができるけど）。DoGA L1/L2はメカモノのモデリング，アニメーション制作には絶大な威力を発揮する。3DデータのままDirect3Dに渡すのはつらいが，2Dのゲームでこそ生きるシステムといえるだろう。アマチュアゲームプログラマ必携ツールであると思う。たいていの人にとっては，ゲーム自体を作るよりキャラクターパターンを描くほうが難しいと思われるのだから。

次回は今回のエンジンをもっと洗練されたものにするために，さらなる深みに踏み込んでみたい。例の高速化（？）のおかげで，DirectDrawでかなりのことができるだろうということがわかったので，遠慮なくいろいろと機能を追加できる。現状でも背景の自由度が低いことを除けば，それなりのゲームを構築することは十分可能だろう。VBでいままで描画速度の不足に悩んでいた向きにはぜひとも活用してもらいたい。

リスト4　サンプル

```
Option Explicit

Public sp As Integer
Public Hitrect1 As RECT
Public i As Integer
Public tLast As Single
Public tNow As Single

Sub GameInit()

With Hitrect1
    .Left = 5
    .Right = 27
    .Top = 5
    .Bottom = 27
End With

Form1.Caption = "サンプル"

SP_init

Rem パターン定義 ************************
SP_def 0, "myship.spr", 32
SP_def 1, "enemy01.spr", 32

Rem スプライト定義 ************************
Rem スプライト番号, パターン番号, 大きさ (x,y), 当たり判定エリア
SP_SET (0), 0, 32, 32, Hitrect1
SP_SET (1), 1, 32, 32, Hitrect1
SP_SET (2), 0, 32, 32, Hitrect1
SP_SET (3), 1, 32, 32, Hitrect1
SP_SET (4), 1, 32, 32, Hitrect1
SP_SET (5), 1, 32, 32, Hitrect1
SP_SET (6), 1, 32, 32, Hitrect1

Rem 背景定義 **************************
MaxBG = 2
BG1.filename = "star1.bg"
BG1.Width = 640
BG1.Height = 480

BG2.filename = "star1.bg"
BG2.Width = 640
BG2.Height = 480

InitBGSurfaces

BG2x = 200
BG2y = 200

Rem サウンド設定************************
LoadSound 0, "tuta.wav"
LoadSound 1, "tutb.wav"

LoadMIDI "winter.mid"

PlayMusic

End Sub


Sub DoFrame()
' On Local Error Resume Next
    ' 以下ゲームメインルーチン************************

    Static X As Long
    Static Y As Long
    Static B As Integer
    Static q As Integer
    Static A As Single
    Static t1 As Single
    Static t2 As Single

    BltStart

Rem キー入力 (カーソルキー＆スペースバー) ********************

    Form1.Label3.Caption = "Stick=" + Str(Stick) + "  Trig=" + Str(DItrig0)

Rem X,Y座標計算********************************

    t2 = Timer
    If t1 <> 0 Then
        A = A + (t2 - t1) * 100
        If A > 360 Then A = A - 360: B = 1 Else B = 0
    End If
    t1 = t2

    X = Cos((A / 360) * 2 * 3.141) * scXsize / 6
    Y = Sin((A / 360) * 2 * 3.141) * scYsize / 6
    X = X + scXsize / 2
    Y = Y + scYsize / 2

    If BG1x > BG1.Width Then BG1x = 0 Else BG1x = BG1x + 2
    If BG2x > BG2.Width Then BG2x = 0 Else BG2x = BG2x + 4


    BGhome

    calculateFPS

Rem アニメーションパターンの例**************************

    XSP(0).Index = XSP(0).Index + 1
    If XSP(0).Index + 1 >= XSP(0).MaxIndex Then XSP(0).Index = 0
    XSP(1).Index = XSP(1).Index + 1
    If XSP(1).Index + 1 >= XSP(1).MaxIndex Then XSP(1).Index = 0

Rem スプライトの配置 *****************************

   SP_MOVE X + 100, Y + 50, 0
   SP_MOVE 500 - X, Y + 50, 1

Rem 当たり判定 *******************************

    If HitCheckSP(0, 1) Then
        Form1.Label2.Caption = "Hit!"
        If Not DSound(1) Is Nothing Then DSound(1).Play DSBPLAY_DEFAULT

    Else
        Form1.Label2.Caption = ""
    End If


Rem 以下直接BLTの例 ******************************

    BltSP X, Y, 2
    BltSP X / 5 + 340, Y * 2 - 200, 2
    BltSP X - 100, Y, 3
    BltSP 0, 0, 2
skip:
        Flip

End Sub

Sub calculateFPS()
    i = i + 1
    If i = 30 Then
        tNow = Timer
        If tNow <> tLast Then
            fps = 30 / (Timer - tLast)
            tLast = Timer
            i = 0
            Form1.Label1.Caption = "Frames per Second =" + Format$(fps, "#.0")
        End If
    End If

End Sub
```

ネットワーク対応3Dゲーム

GALAXY-KNIGHTSの制作(前編)

サイバーヘッド
吉村 功成
Ilustration :
Visual House シブミ
新田 祐康
平舟 勝典

ハードウェアの最適化によって，現在もっとも柔軟で強力なゲーム環境を実現するDirectX。その機能を使ってネットワーク対応3Dシューティングゲームを作っていきましょう。今回は企画段階から始めてゲームのもっとも基本部分となるスケルトンプログラムを作っていきます。解説はサイバーヘッド(旧名アルシス・ソフトウェア)の吉村氏です。

いまを去ること14年前の1985年11月，長崎県の佐世保で，ひと握りの男たちが夢を創る会社を興しました。会社の名前は有限会社アルシス・ソフトウェア。この名前は「ARS・MAGNA」……神話に出てくる「自動機械」……あるいは古語の芸術/創るという意味を持った「ARS」をもとに，現代のコンピュータに不可欠な要素である「SYSTEM」を付加して，「ARSYS」と名づけ，ソフト会社の職人気質=「粋」を忘れないために「SOFTWARE」をつけて作り上げた名前でした。この名を借りて「主張を忘れないソフト作り」に専念してきた彼らでしたが，時代の流れとともに，表舞台に名前が出ることが少なくなり，一度はその存在が消えたとも思われてしまったようです。

しかし，彼らはその間も活動を続け，時代の変遷とともにコンシューマ相手の，ときには自社で，ときには他社の裏方としてですが，時代を確実に支えてきたのです。そしていま再び，新たな名前とともにその人間たちは，独自の活動を始めようとしています。その人間たち=我々の新しい名前はサイバーヘッド。「ソフトウェア」にとらわれない，最先端の仕事を目指そうという新たな目標を形にした名前です。

今回，このOh!X誌上にて，サイバーヘッドが法人として初めて，雑誌という頒布形態で「記事」という，新たなソフトウェアを発表する機会をいただくことができました。この企画を通じて，新たなソフトウェアの可能性を提示させていただけたら，と考えております。

なにとぞ，おつきあいのほど，よろしくお願いいたします。

第1部 どこまでやれるのか?

DirectX7の登場

1999年7月，Windowsのゲーム制作に，ちょっとしたトピックが流れました。それは Meltdown Tokyo1999。マイクロソフト(株)(以下MSKK)が，毎年行っているマルチメディア制作のためのプレゼンテーションです。これは，米Microsoftが行っている Meltdownというプレゼンテーションの日本でのセッションでした(編注:1999年のMeltdownはシアトル，東京，ロンドンで開催されました)。ここで，MSKKは，Windowsマシンでの新しいゲーム制作のプラットホームとして，かねてから発表を約束していたDirectX(ダイレクトエックス)の最新バージョン，DirectX7の全貌を発表しました。

ここで，DirectXについて簡単に解説いたしましょう。Oh!X読者の諸兄には，釈迦に説法とは存じますが，ここはお約束と免じて，ご拝読ください。

DirectXとは，米マイクロソフトがWindowsにおいてのゲームアプリケーション開発のために世に送り出したゲーム開発用のライブラリセット+開発キットの総称です。これは，もともとのWindows開発セットが持っているAPI(ユーザーが作ったプログラムと，OSがやり取りをするためのライブラリ)が，ビジネスやシステムアプリケーションのための，汎用な=冗長なものになっているため，高速な描画や，高性能な音声表現を要求されるゲームアプリケーションには対応が困難なため，専用のAPIが求められた結果でした。

DirectX以前にも，WinGを代表とするいくつかのゲーム専用APIが発表されはしたのですが，使い勝手の面でいまひとつだったため，普及にはほど遠い状態だったのです。DirectXは，ハードウェアを，その名の示すとおり「ダイレクトに」近い状態でアクセスするためのさまざまな方法を，シンプルな関数で提供してくれ，しかも，多数のベンダーから発表されているグラフィックカード，サウンドカードなど，多様なATマシンのハードの差を吸収する統合プラットホームになっていました。

これは凄いことです。なにが凄いかというと，いまのWindowsマシンの性能差を吸収するというのはとんでもないことなのです。グラフィックカードを例にとると，第一に色数と解像度の問題があります。それと搭載メモリの量や，描画のためのアシスト機能，たとえば「画面に線を引く」とか，「画面の領域をコピーする」，「裏画面描画でちらつきをなくす」果ては「3Dの描画を行う」，「光源計算をする」，「半透明の重ね合わせ」といった機能は，すべてのハードで共通にサポートされているわけではありません。そういった性能の差を吸収するには，「どのハードでも持っている，いちばん下のラインにあわせる」という，いわば最低の妥協案が「共通化したソフトウェア」を作る場合の一般常識でした。

しかし，DirectXは，共通化を果たしたうえで，「それぞれのハードでの最高の性能を引き出す」という，ゲームではどうしても必要となる課題を(100%とはいえないにしても)，見事に実現しているのです。

つまり，5年前の旧式ハードであろうが，昨日発売されたばかりの最新ハードであろうが，パフォーマンスの差こそあれ(というか，パフォーマンスの差が重要なのですが)，なんとか動いてしまうのです。いったいどうやっているんでしょう?

秘密は，DirectXの成り立ちにあります。

DirectXは，大きく分けて2つの方法でハードウェアにアクセスします。ひとつはHALと呼ばれ，もうひとつはHELと呼ばれます。

順に説明しましょう。

HAL……といっても，遙か昔，MZ-80Kなどにハードを作っていた某研究所でも，はたまた来年に木星くんだりまで飛んでいく宇宙船で叛乱を起こすコンピュータのことでもありません(まあ問題を起こすという意味では同じようなものかもしれませんが，その点については後述します)。

HALとは，Hardware Abstraction Layer(ハードウェア抽象化層)の

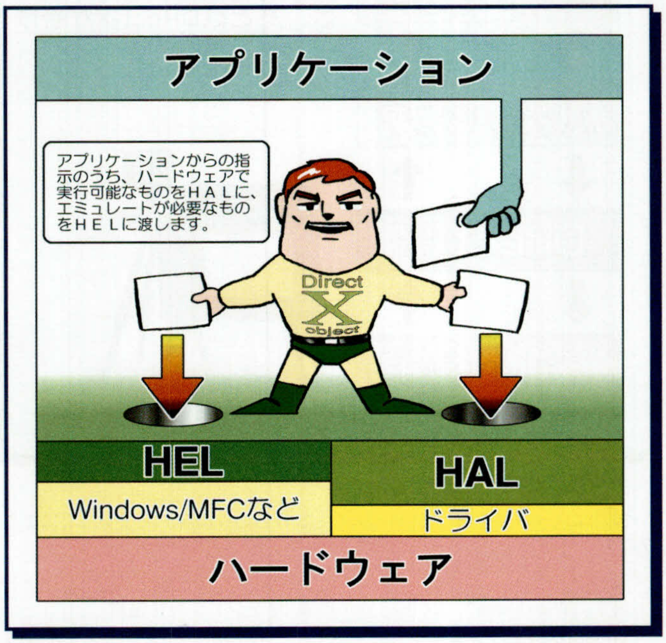
略で，ハードウェアを直接アクセスするための機能を示します。DirectXでは，Windowsマシンに実装されたハードに「どんな性能があるのか」をチェックすることができます。そして，たとえば「テクスチャを貼ったポリゴンを描く」という性能があれば，HALを通じて直接その機能を呼び出すことができるのです。

じゃあ，チェックした機能がないときはどうすればいいんでしょうか？ そこで登場するのがHELです。HELをMicrosoft Bookshelfで調べると（げえ，そんなもの使っているのかとかいう突っ込みはこの際抜きにして），暗黒と死者の国の女神とあります。マイクロソフトの技術者がこのことを意識したかどうかわかりませんが，HELは，HALというプラットホームに見捨てられたいわば死にかけた処理を救済する女神のようなものです。

具体的には，Hardware Emulation Layer といい，ハードウェアを「真似る」ソフトウェアのライブラリ群を指します。つまり，「ハードが載っていないなら，とりあえずソフトでなんとかしちゃおう」ライブラリなのです。

もちろん問題はあります。ハードウェアがないということは，すなわちハードウェアが勝手にやってくれていた内容を，「CPUが全部肩代わりする」ということで，当然，専用ハードが載っているときのような性能は出ません。

というか，「地獄（HELL）」のように遅くなります（案外ここらへんに引っ掛けている気がします）。しかし，ある性能を持っていないからといって，ソフトが動かなくなるという最悪の事態は避けられるわけで，これによって，DirectXを使って作られたソフトの動作は，どんなハードでも，ある程度保証されるのです。



しかし，世の中には万能というものはなかなかないらしく，このDirectXにも当然問題もあります。まず第一は，HALとHELの使い分けの問題です。どういうことかというと，HALに問い合わせをするとき，当然要求するハードウェアを，アプリケーションから指定するわけですが，そのとき，

画面が1600×1200ドットの32ビットカラーで，Zバッファとバリバリのテクスチャもバンプマップも使った3Dポリゴンが使いたい

という風な要求をしたら，さすがに使えるハードは限定されてしまいます。

かといって，じゃあ，最低の性能でいいや，というのでは，せっかくのDirectXの意味がなくなってしまいます。

どうすればよいかというと，DirectXでプログラムを作るときは，

「よい性能のものから順に調べていって，なければ性能を落として再度調べる」

ということを繰り返すことになります。

で，それでもどうしても実現できないとき，初めてHELを使えばよいのです。こういった「性能の列挙」はDirectXのプログラムで，ハードを初期化するときの「お約束」みたいなものです。

残念なことに，このために，「初期化」という最初にやるべき仕事が，DirectXのプログラミングでは，

「いちばん面倒くさくて難解」

ということになってしまっているため，人によってはここで挫折してしまう場合もあるようです。

今回のDirectX 7の登場で，初期化の手順がかなり効率化されたのですが，それでもやっぱりちょっとばかり高いハードルになっているようです。まあ，多少の困難なくして，達成感はないといいますし，いまのところは我慢してこの「狭き門」をクリアしていくしかないのでしょう。

第2の問題は，HAL自体にあります。HAL制作は，実はマイクロソフトの仕事ではありません。HAL層の内容を決めているのは彼らですが，HALを実際に作って実装しているのはハードを出しているベンダーの仕事なのです。ハードをインストールしたときにお馴染みの「ドライバのインストール」という奴がありますが，あの『ドライバ』の中に，HALの機能は入っているのです。

「仕事が分散されるのはよいことじゃないか」

というご意見もあると思いますが，事態はマイクロソフトの妙な（？）頑張りのおかげで，かなりややこしくなっているのです。

DirectXが最初にリリースされたのは1995年9月（日本語版は翌年3月）です。ところが，このあと矢継ぎ早に，DirectXはバージョンアップを繰り返していきました。1996年6月にはDirectX2，9月にはDirectX3，翌1997年7月にはDirectX5（4はリリースされなかった），1998年の6月にはDirectX6（このあと6.1というバージョンもありました）。そして，1999年7月のDirectX7の登場，と，3まではなんと3カ月，それ以降も，年1回のペースで更新されているのです。当然，DirectXの進化に追いつくべく，ハードウェアベンダーは，このリリース時期にあわせてドライバを開発せねばなりません。

一般に，開発を急がせたプログラムはどうなってしまうでしょうか？ 十分なチェック時間も設けられずにバグを内在させてしまうのが関の山です。ご多分に漏れず，ベンダーから供給されるドライバは，DirectXの進化に追いつくのが精一杯で，あちこちに欠損があり，バグの温床と化しているのが現状です。

それでも，近年のDirectXのバージョンアップ（6.0以降）が改良中心になってきているため，若干の余裕ができたのか，最近はかなり状況が改善されてきてはいますが，依然バグは残っており，「あるハードでは動くが，あるハードでは動かない」という，DirectXの思想上，本来あってはならないことが現実に起きてしまっています。

宇宙船ディスカバリーの中で責任の重さに押しつぶされて「ひなぎく」を歌う羽目にならなかったとしても，HALは問題を起こしてしまっているのです。

こういった問題の解決は，本来はドライバ制作側のベンダーの対応を待ちたいところですが，それでは商売になりません。情けない話ですが，「動作を保証しているはずの」DirectXで，多くのメーカーが動作チェックリストを公開しているのが現状です。

まあ，かなりの場合は「動かない」わけではなく，「問題が発生する」レベ

ルだというのが，救いといえばいえるでしょう。現代のバベルの塔AT機に遣わされたエスペラント（死語？），DirectXは，このように，完全とはいい切れないものですが，Windowsでのゲーム制作の切り札であることもまた事実です。

我々アプリケーション制作サイドでは，マイクロソフトの今後の動向に注目し，この不安定かつ強力な「暴れ馬」API，DirectXを上手く乗りこなし，一流のコンピュータカウボーイとして名を馳せるべく，日々精進するしかないでしょう。

前置きがずいぶん長くなってしまいました。ぼちぼち本題へ進みましょう。

1999年夏，新しく発表されたDirectX7は，従来のDirectXに比べ，

パフォーマンスの向上
ステレオ表示サポート
3D曲面のサポート強化
その他のインタフェイスの強化
VisualBasicの正式サポート

そしてなにより，

初期化手順などの，関数の大幅な単純化

が図られているのが最大の特徴といえるでしょう。

巷で数多く見かけるDirectX7についての記事は，残念ながらこの中の「VisualBasic対応」の部分ばかりクローズアップして取り上げています。確かに，VBで絵がガシガシ動いて，音楽もバリバリのソフトが作れるというのは，とても画期的で素晴らしいことです。

しかし，プロダクツ制作サイドからすると，せっかくパフォーマンスが向上したDirectXを，結果的なコードの速度の点で劣ってしまうVBだけに使うのはもったいないですし，関数の単純化が図られたおかげで，VisualC++を使った開発も簡単になったのですから，その真髄をしゃぶり尽くしてみましょう。

●3Dでマップ表示（Direct3D）

多くのDirectXの解説本や解説記事が，最初に2D描画のためのエンジンであるDirectDrawを取り上げています。

これは，Direct3Dを使ううえではDirectDrawの機能を避けて通れないことと，WindowsGDI（Windows標準のグラフィックドライバ）と，DirectXの接点や違いなどを見せるためにもっとも適しているからです。本記事ではあえてこの慣例に倣わず，Direct3DとDirectDrawを並列で取り上げることにします。そのほうが，「描画」という一連のパイプラインを一気に説明できるため，雑誌記事としては都合がよいと判断したためです。また，DirectX7から新しく登場したD3DXユーティリティライブラリで，DirectDrawとDirect3Dを簡単に共存して扱えるため，もはや，旧来の2D→3Dという考え方がなくても，ある程度プログラミングできてしまうという事実もあるためです。

もし，この濃縮状態の記事ではつらい，とのご要望がありましたら，記事末に関連情報のサイト，参考文献などをまとめますので，参考にしてください。

では，さっそく参りましょうか。

DirectXでは，すべての操作はオブジェクトに対して行われます。3D描画するためのデータも当然オブジェクトですが，3D画面もオブジェクト，そして，それを受けとめる表示画面（実際はDirectDraw）もオブジェクトです。オブジェクト指向プログラミングがわからないと，DirectXを使うのはかなりつらいということにもなります。オブジェクトとは，いってみればある作業を一括して行う作業員のようなものです。

たとえば，DirectDrawオブジェクトに，「画面をクリアしてくれ」と頼めば画面をクリアしてくれ，「このビットマップを表示してくれ」と頼めば表示してくれるわけです。よくいわれる比喩ですが，あまりに抽象的すぎてわかりにくく，ここでつまずいてしまう人もいるようです。

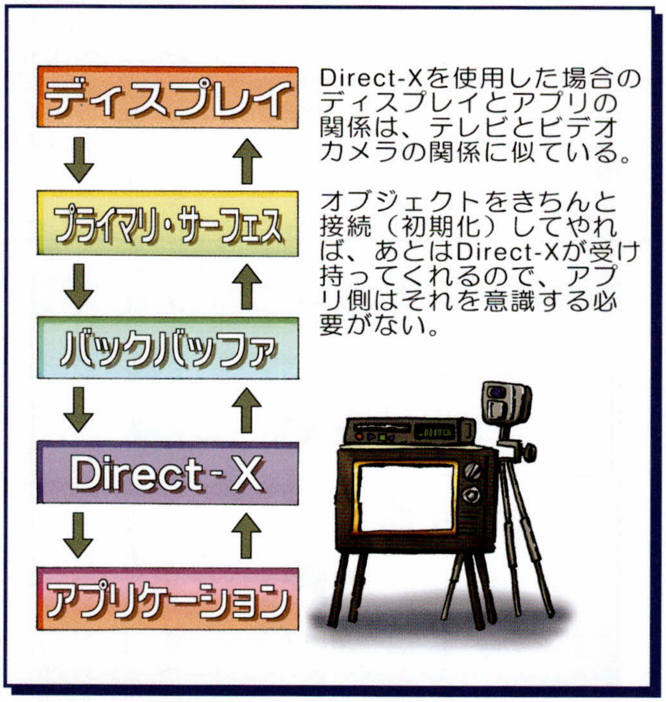
オブジェクトは，別の考え方だと，「コンポ」や，「相互接続して使う電化製品」のようなものだともいうことができます。

たとえば，テレビとビデオ，それとビデオカメラのようなものです。それぞれの機械がそれぞれの機能を持ち，単体でも，とりあえず独立して動く性能は持っています。

それが，細いケーブルで接続するだけで，お互いが通信しあって，単体ではできなかった「ビデオ編集」という機能を持つようになるのです。DirectXオブジェクトも似たようなもので，「アプリケーション」が提示する物体を，「Direct3D」が受け取り，バックバッファ描画します。

そして，ひとつのシーンを書き上げたら「画面のフリップ」（つまりコマ送り）は「DirectDraw」の出番です。

そうやって「ディスプレイ」に画面を出していきます。



実際にDirectXを使って3D描画をしようという場合，大まかに次にような手順となります。

- まず，Windowsですから，Windowオブジェクトを生成
- DirectDrawオブジェクトを生成して，Windowオブジェクトに登録
- フレームバッファオブジェクトを作成して，DirectDrawオブジェクトに登録
- ダブルバッファでちらつかない表示をしたければ，バックバッファオブジェクトを生成して，フレームバッファオブジェクトに登録
- 3Dの隠れ面処理用にZバッファオブジェクトを生成して，各バッファに登録

……従来はこうでした。

ところが，です。D3DXの登場で，すべては簡単になってしまいました。

- まず，Windowオブジェクトを生成（これはさすがに同じ）
- D3DXライブラリの初期化

・D3DXコンテキストオブジェクトの取得（従来のDirectDrawとDirect3Dがこれだけで使えるようになります！）

これで準備が終わりです。

簡単になったのは準備だけではありません。実際の描画サイクルも至って簡単です。

・画面クリア（ID3DXContext::Clearなど）
・描画コマンド発行（ここは従来と同じ。もちろんすでにDirectDrawとDirect3Dが混在して使える！）
・画面更新（ID3DXContext::UpdateFrameを使用）

なかなかとんでもないのは最後の画面更新です。

一般に従来のDirectXのプログラムはフルスクリーン（ゲームがスクリーンを占有）と，ウィンドウ（DirectXでの動作も，ほかのアプリケーションと同じく1個のウィンドウになる）では，異なるアプローチが必要でした。しかし，賢いD3DXライブラリのおかげで，この差は吸収され，ほとんどいじることなく両者を両立できてしまうのです。

このD3DXユーティリティライブラリを使えば，どれだけ簡単にアプリケーションを作れるか，についての詳細は，この記事の第3部にてサンプルを作って検証します。

いまの時点では，「D3DXという素敵なおまじないがある」と考えておいてください。

通信対戦だ！

描画の話はこれくらいにして，簡単に通信についても触れておきましょう。これも，今回の第3部において，サンプルで具体的に扱います。DirectXでは，通信を行う部分のことをDirectPlay（ダイレクトプレイ）と呼びます。

このDirectPlayは，簡単にいうと，Windowsが作れるあらゆるタイプのネットワークを利用して，ほかのプレイヤーと交信するためのライブラリです。あらゆるタイプ……というと曖昧ですが，ここらへんは実は先ほどまで解説したDirectDrawなどの場合とまったく同じで，接続できるデバイスと，その性能によって，それぞれがある種の「通信装置」のようなものである，と考え，いま使えるタイプの通信装置をDirectPlayが見つけてくれるのです。そのなかから，使いたい装置を選択すれば，任意のアクセス方法が実現できるというわけです。

ちなみに，この「通信装置」は，DirectPlayでは，「サービスプロバイダ」と呼ばれます。皆さんがインターネットを使われるとき「インターネットサービスプロバイダ」と呼ばれる業者を使いますよね？

俺は直接基幹網から専用線引いてるぞ，とかいうリッチな方はこの際シカトです……というか，その方自体が「サービスプロバイダ」なのです。日本語ではよく「接続業者」と訳されていますが，まさに「接続を担ってくれるもの」のことを示します。

DirectPlayで使う「サービスプロバイダ」も，その意味ではまさに同じで，たとえば「TCP/IPで接続したい」のか，「IPXプロトコルで接続したい」のか，はたまた「シリアル回線で直結したい」のか，など，通信方法はいろいろあると思いますが，DirectPlayでは，どれか使いたい通信方法を選択するだけで，接続方法を自由に切り換えることができます。

またしてもDirectXお家芸の「ハードの差異を吸収」をやってのけてくれるわけです。あとは，接続形態さえちゃんと押さえていれば，ネット対戦ゲームも自由自在です。

……で，問題なのはここからです。

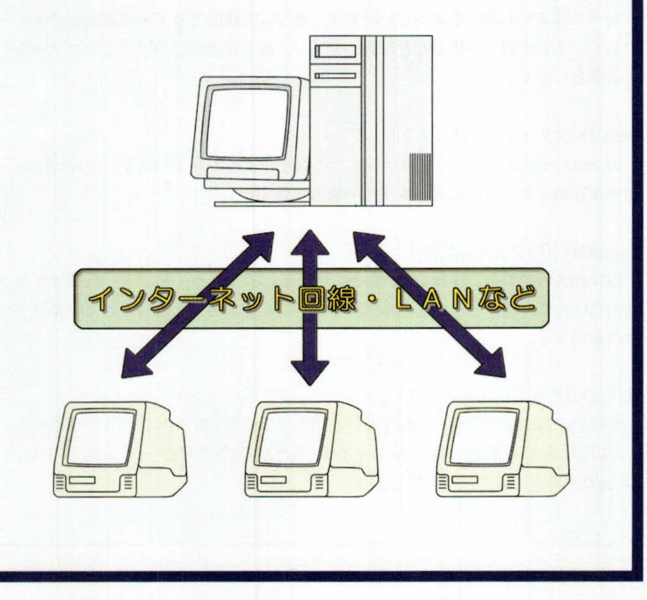
ゲームをする場合，通信方法は大きく分けて2通りになります。ひとつは「サーバ&クライアント形式」と呼ばれるもので，1台の「サーバマシン」に，たくさんの「クライアントマシン」を接続する形態のものです。

この形式の利点は，「サーバマシン」に主要なデータを置くことで，「クライアントマシン」の負担を減らすことができる点と，サーバマシンにユーザー情報を置くことで，たとえばサーバ中にある街に「自分の家を建てたり」，「ランキングに名前が残る」といった「存在し続ける街」を作ることができる点でしょう。この形式の代表的なものは，かのUltimaOnline（以下，UO）です。

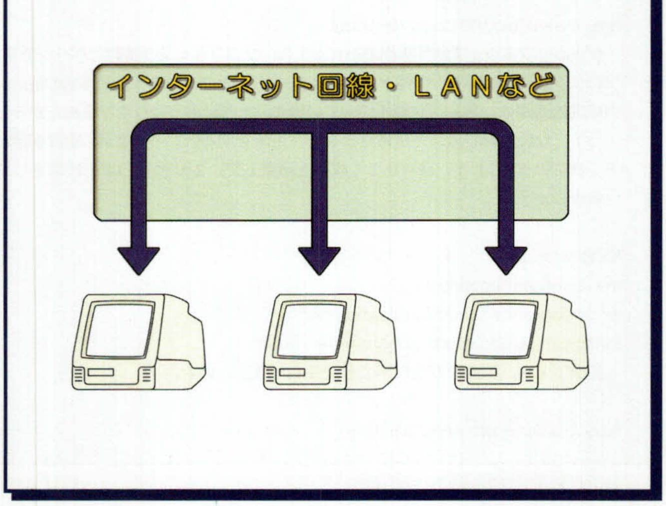
ただし，この形式にはちょっとした欠点があります。それは，「サービスを停止しない限り絶対に電源を切らない」マシンが必ず1台以上必要になるという点です。これは，たとえば企業がサーバを立ち上げてゲームサービスをする場合はよいですが，個人が好きなときに仲間内でゲームをする，といったシーンでは，実現がかなり困難なものになります。

そこで登場するのが「ピアtoピア形式」と呼ばれる方式です。この形式では，「サーバ」と呼ばれるマシンは存在しません。いうなれば，「隠れんぼするもの，この指止まれ」と，最初に呼びかけるプレイヤーがいて，このプレイヤーが「セッション」といわれるゲームを立ち上げるのです。

サーバ&クライアント型接続



ピアtoピア型接続



いったんゲームが始まり，複数の人間がセッションに入ってしまうと，この最初に呼びかけたプレイヤーは「やーめた」といって，セッションから抜けてしまうことができます。しかし，そのセッションにひとり以上のプレイヤーがいる限りは，セッションは存続し続けます。ちょうど，仲間同士で遊びに行くとき，最初に誘った奴が先に帰ってしまうようなもの，と考えればわかりやすいでしょうか。

最後に残ったメンバーが「お開き」というまでは，遊びは続くわけです。この形式の利点は，なんといってもサーバマシンが不要なこと，広域ネットワークでなくても，たとえばシリアルケーブルで繋がっている2台のマシンでも，問題なく遊べるという点が挙げられるでしょう。

ところが，ローカルに強いピアtoピアですが，広域ネットワークでは厄介な問題が待っています。なにかというと，サーバのような「指標」がないため，見知らぬ人同士ではゲームを始めることができない，という点です。

要するに，知っているマシン同士なら接続できますが，知らないマシンでゲームが立ち上がっていても，そこに接続するわけにはいかない。ということです（実際は，TCP/IPなら，ブロードキャストの及ぶ範囲なら，無差別に相手を探すことも可能ですが，実際問題としては実用にはほど遠いでしょう）。

サーバクライアント形式ほど本格的なサーバはいらないけれど，仲間を紹介してくれる場所があるとよいな。そんな考えが浮かんでくるのは当然といえるでしょう。

そこで登場するのが「ロビーサーバ」です。

これは，「ロビー」といわれる，ある種のチャットルームを用意して，そこに集まった人が「ピアtoピア」でゲームを始める。という，サーバクライアントと，ピアtoピアの折衷案のようなものです。

利点といえば，サーバに負担がかからないため，割と設置が楽なこと。そして，いったんゲームが始まってしまっていても，ロビーから，どんなゲームが現在進行中か，というセッション表を呼び出すことができるので，セッションに途中参加するのも簡単ということがいえるでしょう。

本記事では，今回は実験ということで「ピアtoピア」にて始めますが，最終的には再配布可能な「ロビーサーバ」もセットとして，プロダクツをまとめていきます。



ロビーサーバ同士が繋がる機構さえ確立すれば，巨大なサーバを持つUOにすら対抗できます。これは，ロビーサーバゲームの代表格「Quake」シリーズが実証しています。最終目標はインターネットを網羅するゲーム網。なんて，格好いいと思いませんか？

Column

●DirectX 関連情報サイト

http://www.microsoft.com/japan/directx/

ご本家MSKKのDirectX公式ホームページです。DirectX7の開発キットのダウンロードもできますが，この記事とともに同梱されるはずですし，なにより馬鹿でかい（128MB!）ので，やめておいたほうが賢明です。DirectXでのプログラミングに関する資料（英文がほとんどですが……）が手に入ります。本家の正式な情報源ですので，絶えずチェックするのが基本でしょう。

http://www.bio100.co.jp/dev.html

ゲームソフトウェア制作集団:Bio100％の，プログラミング情報のページです。プログラミングに関する情報や，DirectX-BBSなどがあります。日本のDirectXの現実的活用は，彼らが先陣を切っています。ただ，あとからやり始めたからといって，なにも恥じることはありません。先陣を切るということは「地雷を踏んでくれている」のです。ありがたく成果を頂戴して，より効率のよい仕事をしていきましょう。

NEWS-Groups
microsoft.public.directx
microsoft.public.multimedia.directx.*
microsoft.public.win32.programmer.directx

英文ですが，開発者が生の声で意見を戦わせています。

fj.os.ms-windows.programming

日本語です。DirectXだけではなく，Windows全般のNGですが，DirectXについての議論も頻繁に行われています。

書籍

DirectX実践プログラミング

ソースリストと細かく対応した記事は，確かに実践的でとても直感的にできています。その分細かい解説などは抜いてあり，ある程度のプログラミングスキルを必要とします。

DirectXグラフィックプログラミング

WindowsGDIを使ったプログラムとの対比がよくできています。VisualC++自体が初めてという方にもおすすめできる1冊です。

Direct3Dプログラミングガイドブック

DirectXの中でも，Direct3D中心に話をまとめている1冊です。記事はどちらかといえば，読みもの中心の構成になっています。付録に添付されているソフトが有用です。

DirectX6.1公式ハンドブック

分厚い，高い本ですが，これがないと始まりません。MSKKのサイトがもっと和訳に力を入れていれば，なんとかなると思うのですが……。もうすぐ，DirectX7の本も発売になるでしょう。

第2部 ゲームシステムを決めよう

時代はネットワーク対応

インターネットの普及で，ネットワークというものもずいぶん浸透してきたように思います。実際，UOを頂点として，Quake，ディアブロなど，海外のオンラインゲームは次々と進化を重ねています。それにひきかえ，日本国産のネットワークゲームはどうでしょう。お世辞にも流行っているとはいいがたい状況です。一般には，この差を生んでいる原因は，ネットワーク使用料の問題が第一だといわれています。

確かに，日本では，まともなネットワークゲームをやろうとすると，テレホーダイなどの割引サービスを使っていない場合，5600円くらいは，あっという間にかかってしまいます。月間ではプロバイダ料金込みで2万円などというのも当たり前です。

しかも，回線の質は，よくてISDNバルク転送の128kbps，ほとんどの人はいまだに28kbps程度のモデムを使っているのが実情のようです。これに対して，アメリカ，西欧では，ケーブルモデム（512kbps）や高速なADSLモデム（1M～8Mbps）が，主流になりつつあります。ああ，海外はいいなぁ……。

しかし，ちょっと待ってください。流行っていないのは「国産」ネットゲームなのです。日本にも，海外のネットゲームには根強いファンがたくさんいます。コンシューマゲーム機と比較すると微々たるものですが，かつてパソコンゲームで「怪物」と評されたXanaduでさえ，17万本だったのです。そうでなくても，パソコンソフトは1万本を超えれば大ヒットでした。

それを考えたら，いまの「海外産」ネットゲームには，1万本をクリアした作品はたくさんあります。言葉や文化のギャップを超えて売れているわけで，十分に健闘しているといえるでしょう。対して，日本純国産のネットゲームは，存在はしていますが，ほとんど騒がれることもありません。

なぜでしょう。

まず，「日本で，PCでゲームを作ろうとする会社が少ない」ことが挙げられます。それは，「PCでゲームを作っても売れない」という，「コンシューマゲーム機中心」になってしまった日本のゲーム生産の現状があるからです。いまでも残っているのは，コンシューマゲーム機では疎まれるような「馬鹿でかいメモリを要求する」ゲーム，「18歳未満お断り」ゲームや，「画面が地味」なゲームなのです。

上記のどのゲームをとっても，ネットワークという環境には不向きだということがいえます。馬鹿でかいメモリを前提としたゲームには，いまのネットワークの速度は遅すぎます。18禁ゲームでは，まず「ネットワークで対戦」という考え方と結びつきにくいでしょう。画面が地味なゲームは実際にかなりの数がネット対応になっていますが，それ自体があまりぱっとしないのが実情です。

日本ではネットワークゲームの制作は無理なのでしょうか？　いいえ，そうではないようです。答えは割と簡単なところにありました。近年の携帯電話，特にメール機能を持った携帯電話の普及速度は目を見張るものがあります。「メル友」とか，「出会い」などのキーワードを基にしたホームページは桁違いのヒット数を誇っています。

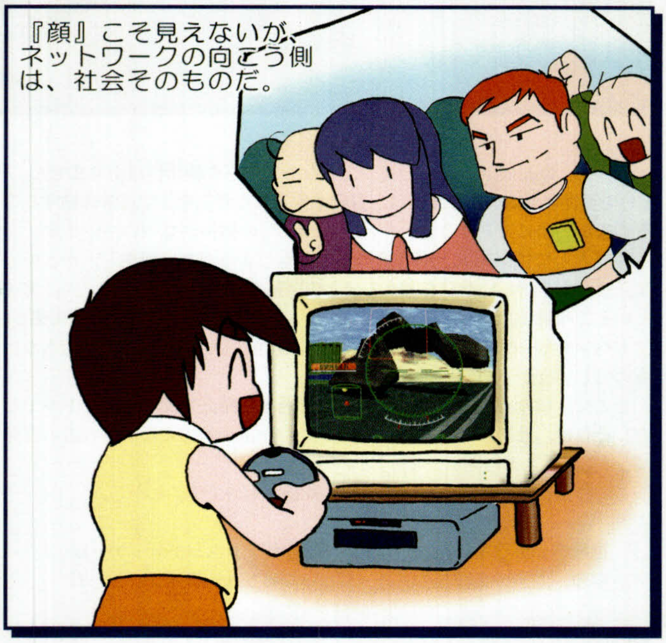
キーボードより遥かに操作しにくい携帯電話で，セコセコセコセコ，一所懸命メールをやり取りしている女の子たちの姿が，街のあちこちで見受けられるようになってきました。メールや携帯電話は，彼ら，彼女らにとっては，ゲーム以上に刺激的な「遊び」なのです。そこには，自分の正体を隠して相手とやり取りする，仮面舞踏会のような楽しさがあります。知らない人たちと偶然の出会いをするスリルがあります。知っている相手にメールを送るときでさえ，面と向かってしゃべっている自分とは別の自分を演出した気になります。携帯電話は持っている人間をネットの中にいる別人に「変身させる」ツールなのです。

しかもそのツールの通信能力たるや，370万台を売り上げたiモードでさえ，ひと昔前のパソコン通信と同じ，9600bpsなのです。しかも料金の問題はいわずもがなです。つまり，国産ネットワークゲームは，インフラが悪いから普及しないのでも，通信費が高いから普及しないのでもありません。「売れるネットワークゲームを作らなかっただけ」だということはもはや明らかでしょう。

海外ネットワークゲームの名作をよく見てみると，UOやQuakeシリーズは，いずれも「変身願望」や「覆面性」を，実にうまく使っていることがわかります。まず，UOでは，実際に別人として「生活」できます。財を成し，あるいはイベントをこなして，UO世界での有名人として名を馳せることも不可能ではありません。

Quakeでは，誰ともわからない相手と，銃撃戦で生き残りを競います。日常の自分を脱ぎ捨て，欲望の赴くままに戦う，別の自分がそこにはいます。

ゲーム世界でのこうした「変身ごっこ」は，オフラインのコンシューマゲーム機では決して味わえない興奮を与えてくれます。なにせ，自分の戦いを「見てくれる人がいる」のですから……。



携帯電話全盛のこの時代，インターネットの普及ともあいまって，ネットワークゲームの時代の素地はできているのです。私たちは今回，ネットワークゲームを作るにあたり上記のことを念頭に置きました。

ゲームに参加する人が変身願望を達成でき，かつ，同時に参加した人々と経験を「共有する」。いわば，観客参加型アトラクションのようなゲームです。

3Dシューティングの敷居が高い？

ネットワーク型ゲームを作るためには，先人の努力を見習うべきでしょう。ネット「対戦」ゲームで一世を風靡しているのは，なんといってもQuakeシリーズ（またはその類似作）をおいてほかにはありません。複雑な3Dダンジョンで，ただひたすらに撃ちまくり，相手が落とした武器や，そこかしこに落ちている武器やアイテムを，誰よりも先に拾い，情け容赦なく敵を撃ち殺す……。そこには，通常では決して経験することのない淡々とした戦いがあります。一瞬一瞬に集中させるそのゲーム性は，プレイヤーのアドレナリンを大量に呼び覚まし，プレイヤーはときには黙々と，またあるときは奇声をあげながら敵を倒し続けるのです。



しかし，QuakeにはUOの持つような，生活感や物語性はありません。UOで展開されるシナリオには，自分が他人になり，そこで通常体験することのない物語に遭遇するという「ごっこ遊び」の精神が息づいています。

後発としては，先人と同じ真似をしてもしょうがないですし，かといって，まったく新しい遊びを考えるほどの余裕もありません。できたら「美味しいところ取り」がしたいものです。シューティングゲームの持つ爽快感と，アドベンチャーやロールプレイングの持つ物語性や，謎解きを融合できないものでしょうか……。

そこで，はたと思いつくのが，弊社で過去に発売していたタイトルでした。それは，最初の作品の発表はすでに12年前のものとなる古い古い作品で，名前を「スタークルーザー」といいました。

この作品自体は，いま見るととても稚拙なものです。しかし，

謎解きシナリオからなる，ちゃんとしたアドベンチャーゲーム
宇宙，地上の２種類のステージからなるシューティングゲーム

の融合という，それまでにないコンセプトをゲームとして成り立つバランスで作り上げたことに関しては，自負を持っている作品でもあります。それだけに，作り上げた時代のハードや，人的資産の制約もあり，思うような作品に仕上がっていない，モノ作りを生業としている私たちには「悔しいできあがり」の作品ではありました。

しかし，現在のハード，そしてソフトの現状は，当時考えていたコンセプトを実現するには十分なものとなっています。どうしてもクリアしなければならない点は，「人的資産」ですが，これは後述する理由からクリアできる状態になってきました。あとは，いまの人々にあわせたプロダクトに仕上げるのが，私たちに課せられた使命といえるでしょう。そこで，今回のこの企画のゲームは，「スタークルーザー」を基本として，これをいまの人々にマッチさせて作ったものとして設定します。

そうそう，いつまでも「このゲーム」でも格好悪いので，コードネームを決めましょう。スタークルーザーは，「星」の「舟」から名づけました。今回は，ネットワークゲームですから「星々」でしょう。とすると，「銀河」あたりが適当でしょう。そして，「乗りものにまつわるお話」を作るのではないため，「舟」といった，モノに限定するのはなんとなく抵抗があるので，ここはプレイヤーのことにしてしまいましょう。戦うゲームですから，「兵士」あたりがいいんですが，ちょっとひねって，「騎士」にしちゃいましょう。

「銀河騎士」＝「GALAXY-KNIGHTS（ギャラクシーナイツ）」
これが今回のゲームの名前とします。略してGKですね。

いまの人々にあわせたプロダクトってなんでしょうか？

よく聞くのは，いまは「ライトユーザーの時代」だという話です。趣味でゲームをやっていて，多少のわかりづらさや，操作の困難さには目をつぶって，あるいは暗黙の了解で，ゲームをしている人々＝コアユーザーの時代は終わり，「操作が簡単」で，「難しいことを考えなくてもいい」，「ノリでできるゲーム」をほしがる，ファッション感覚でゲームをするお手軽感覚の人々＝ライトユーザーがゲームを買う購買層の中心となったという話です。

確かに，操作性の難解さや，よくこなれていないゲームシステムをプレイヤーに押しつけるのは，昔のソフトハウスが犯していた傲慢な過ちだと思います。しかし，たとえ多少難解でも，それが十分面白い内容で，人を虜にできる可能性があるものである場合，「難しいから切り捨てる」という短絡思考は危険だと思います。

いまのゲームは，「面白さ」の部分まで「ライト」になってしまい，薄っぺらになってしまったものが多々あるように見受けます。ゲームのコアとなる部分の面白さはそのままに，「人にやさしい」システムを作る……。

これは，ライトユーザー云々という話を抜きにしても当たり前のことで，この当たり前を怠っていたからこそ，昔のゲームは問題があったのだと思うのです。その最たる例は，シューティングゲームというジャンルに見受けられました。

この種のゲームが受けなくなっていった理由を，私は次の3つの要素に分析しています。

1）特殊な『お約束』による，現実から遊離した設定
2）バランスが一部の人間のみ対象で，万人受けでなくなっていった
3）特殊な状況（アーケード）に特化したゲーム進行

確かに，シューティングゲームには，複雑なルールはなく，ゲーム自身は非常にシンプルです。「誰よりも技術を磨き，敵の弾を避け，敵を倒す」シューティングはこの技を競うゲームです。同程度にシンプルなゲームで，いまだに人気のあるものとしては，レースゲームが挙げられます。「誰よりも技術を磨き，クラッシュせずに速く走る」レースゲームはこの技を競うゲームです。しかし，片や日本有数の売り上げを誇るモンスターソフトとなり，片や滅亡の憂き目に遭っています。この差はなんでしょうか？　レースゲームは，「車」という現実に存在する機械を使い，現実に行われている競技をゲームにしたものにすぎません。いわばシミュレータです。

このため，レースゲームには特殊なお約束はほとんどありません。車の運転とほぼ同じ機構を持つ操作系で，より速く走ればよいのです。しかも，そこに出てくるマシンは近年のコンピュータの表現力の向上により，現実の車が使われ始め，いわゆるゲーム好き以外の「車好き」が，ゲームに目を向けるという，一般人を上手く釣り込む作戦ができたのです。車で速く走れれば自慢できますし，野郎どもの女の子への受けもよいというものです。

しかし，シューティングがいくら上手くても，現実世界では，特に日本の現実世界では，なんの役にも立ちません。せいぜい，蝿叩きや射的が上手くなるくらいでしょうか？　習熟の成果がゲームの世界という特殊な世界に限定されてしまうのでは，いくら習熟しても空しいというものでしょう。

近年の列車ゲー＆ダンスゲー＆音ゲーが受けているのも，現実との接点があるという点は，外すことはできないと思います。

現実との接点がないゲームは受けないのか？　というと，そういうわけではありませんが，習熟という行為が空しくなってしまう一因になっているのは間違いないでしょう。

さらにその習熟度が人によってかなり差が出てしまうというのも問題です。シューティングゲーム最盛期のゲームは，正直いって「ゲームを極めた人のためだけにある」といっても過言ではなかったと思います。一部の「プロ」が出現し「16連射」などという技を披露したがために，ゲームの難易度アップはお祭りのような騒ぎとなり，一般の人が軽く時間を潰す，といった

ニーズをはねつけてしまったのです。

商売において，せっかくあるニーズを切り捨ててしまうのは「阿呆」と同じです。あるニーズに特化した製品ができるのは構いませんが，それは全体のラインアップあってのものでしょう。この結果，「プロ」の衰退とともに，シューティングという分野全体が縮小してしまったのは，自業自得というものでしょう。

また，登場当時からシューティングゲームは，活躍の場の多くが，ゲームセンターという場でした。そのゲームセンターには「インカム」と呼ばれる客の回転による収益性の問題があります。平均５分に１人が１プレイ分100円を投入してくれないと，採算が取れないのです（１日10時〜24時＝14時間稼働するゲームセンターで，

（14×60分）÷５分×100円＝16800円

当時，基板１枚10万〜15万円で，１週間で代金を回収しないと利益が出ないといわれていたので，妥当な線でしょう）。

そこで，ゲームセンターに置かれるゲーム機は，機械にプレイヤーがいない，「遊んでいる」時間も考慮して，約３分目にいきなり難しくなり，ゲームオーバーになるような設定が求められました。俗にいう「インカムによる制約」という奴です。この設定を，シューティングゲームは実に上手にゲームシステムに取り込みました。

ゲームバランスにある程度の波をつけて，３分台に強いボスが登場するなどして，ゲームの難易度を一気に高くしたのです。この設定は大いに受け，収益性もよいゲームということで，ゲームセンターでのシューティングゲームの地位は確立されました。

しかし，それがゲームが「コンシューマゲーム機」という，家庭に持ち込まれたゲーム環境に変わる際，シューティングゲームにとって致命的な裏目に出る結果となってしまったのです。

ゲームセンター時代に培われたインカムのための技術は，いつのまにかシューティングゲームのゲームバランスに深く根を下ろしてしまい，インカムを考えなくともよい，むしろプレイヤーの誰もにエンディングを見てもらいたいコンシューマゲーム機でさえも，難易度を下げてゲームを作るのが困難になっていたのでした。

やがてPlayStationなどの，いわゆるゲーマーではない，一般人を購買層としたゲーム機が主流になってくると，多くの人はマップの謎解きとか，アイテムの集め方など「話題になる」ゲームへとその興味の対象を移していき，「技を磨く」ストイックなゲームであるシューティングゲームは，シミュレーション系のストイックなゲームのような別方面との連携というわけにもいかず，急速にその勢いをなくしていきます。

いまや，ゲームセンターでもガンシューティング以外のシューティングは影を潜めていますし，コンシューマでは，数えるほどしかシューティングのタイトルは出ていません。

では，シューティングはもはや受け入れられなくなってしまったのでしょうか。答えはノーといってしまいましょう。代表として真っ先に例としたQuakeが挙げられます。PlayStationでは，アドベンチャーとの融合例として，バイオハザードや，メタルギア・ソリッドなどがあります。バイオハザードは，パソコンで「アローン・イン・ザ・ダーク」として知られるソフトのゲームシステムに，SFホラーのストーリーを融合させた作品で，ゲームとしては決して目新しいものではなかったのですが，PlayStationでは「かつてないゲーム」だったという点と巧みな演出でセールスを伸ばしました。その他のゲームは，バイオ……に比べると，舞台がフル3Dになり，自由度が上がりました。ゲームの方向性も，各作品で独特になっています。

これらを見ていると，シューティングは決して閉塞したジャンルではなく，「あり方が変わりつつある」ジャンルとでもいうことができるでしょう。

私たちが過去，X68000専用に作ったKnight Armsも，そのひとつです。物語を，一切のメッセージを排して，シューティングのステージの移り変わりだけで表現する。そして，操作感覚から訴えかけるという，もうひとつの「メッセージ」を，「奥行きのある2Dステージ」や「自由に探索できるスクロールステージ」，そして「突き進むイメージの3Dステージ」という流れで表現しました。

だんだん，可能性と希望が見えてきたような気がするでしょう？　よし，シューティングで行きましょう！

……盛り上げておいてなんですが，実は，乗り越えなければならないハードルはもうひとつあります。

シューティングゲームのなかで，特に3Dシューティングは，いちばん厄介なのです。現在発表されているゲームですら，多くの問題点が残っています。これを解決しなければ，意図したことが伝わる以前に，「ゲームにならない」と，投げ出されてしまうことだってありえるのです。

それはいったいどういうことなのでしょうか？　3Dシューティングが優れているところを考えると，

1）臨場感
2）狙える空間の広さ
3）移動の自由度

が挙げられると思います。

しかし，現実には，上の３つをすべて実現しているゲームは少なく，また，実現しているゲームは操作が厄介で，お世辞にも簡単と呼べるものはそう多くありません。

なぜでしょう。

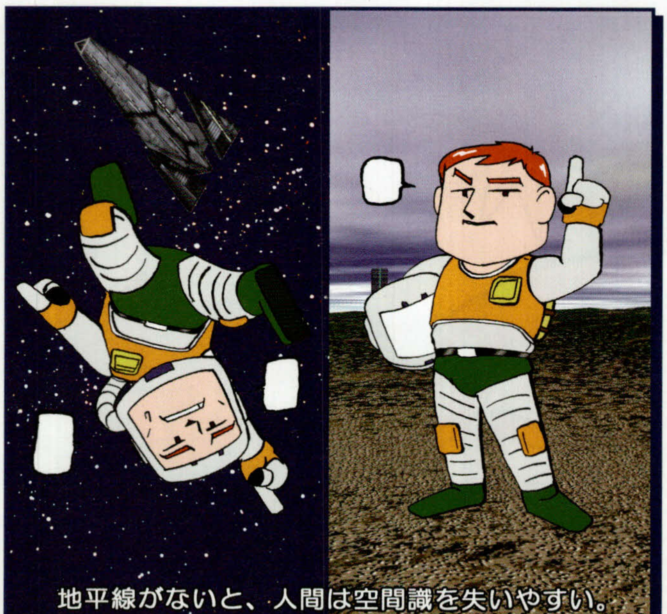
それは，3Dシューティングが，2Dのシューティングと比べて劣っている点に原因があります。

1）空間が広くなった分，認識しにくくなった
2）３軸の移動は，操作が煩雑
3）自由に動けると，敵や目標物を見失いやすい
4）主観視の場合，自分の後ろ，上，下，真横などが見えないため，死角が大きい

すべてに共通しているのは，「複雑になった」ということです。まあ，これは2D→3Dと，軸がひとつ増えたのですから，当然といえば当然ですが，せっかく利点が増えたのに欠点のほうが多くてはお話になりません。どうやって解決したらいいんでしょうか……。

複雑になったのなら，簡単にしてみたらどうでしょう。この場合の複雑さは，大別すると以下の２種類にまとめられます。

①：「認識の複雑さ」



②：「操作の複雑さ」

「認識の複雑さ」の根源的な問題は「自分がいまどこに，どんな姿勢でいるか」でしょう。これがわからなくなると，「自分がどこに向かっているか」とか，「敵はどこか」といった，基本的な認識がすべて駄目になります。人間は現実の3次元空間では，全身感覚としての「重力の方向」と，視覚としての「地平線」を利用して，方向間隔を保っているといわれています。ならば，これを利用しない手はありません。

ゲーム中に，「地平線」を設けてしまいましょう。姿勢が崩れても，この地平線を頼りに態勢を立て直すことができます。

いえ……いっそのこと，体勢を立て直すボタンを作ってしまいましょう。姿勢制御ボタンで，身体は地平線に垂直に立つことにします。

さあ，これで「認識の複雑さ」はかなり軽減されました。さらにこの状態で，地平線上に物体を配置すれば，「自分がいまどこにいるか」を知る手がかりとなります。

次が厄介な「操作の複雑さ」です。たとえば，左斜め後ろ，やや上方向から敵が接近しているとします。照準をあわせるには，いったん地平線方向で旋回したあと，仰角を調整すればよいでしょう。しかし，敵も自分も常に動いているとすると，「ここにいたはず」と思って振り向いても，敵はすでに別のところ……ということは十分にありえます。どうすればよいでしょうか。

最近のゲームでは，「注視」という概念がよく登場しています。「ゼルダの伝説—時のオカリナ—」のZ注視や，「電脳戦隊バーチャロン」の対戦相手を自動注視する機能，「オメガブースト」のスキャンボタン……などがそれです。これらは「サーチすべき物体を自動的に探し，ボタンが有効な間，物体の方向を自動的に見る」というもので，これなら敵の場所を探してうろうろする必要がありません。

11年前の作品であるスタークルーザーですら，これらのご先祖様とでもいうような機能として，マップで敵や目的地を示して，その場所への航路を自動設定するという「オートパイロット」というものを装備していました。自由に空間を移動できるソフトでは，今後これらの機能は不可欠なものとなってくるでしょう。

ただ，やりすぎると別の問題が出ます。いつでも敵を見つけられるため，「敵が背後から来て，追い抜いていく」シチュエーションを作っていても，プレイヤーに見つかってしまうのです。

これだと，せっかく自由に動ける空間を作っても意味がありません。そこで，「注視」にも死角を作り，「一度も視界に捕らえていない敵には注視しない」というルールを導入するなどして，ゲームの流れを創り出す必要があります。

今回のGALAXY-KNIGHTSでは，「注視すべき」というマークを植え付けられたものに対して，注視（以後「サーチ」と呼ぶ）が，サーチボタンを押している間効く，という風にしてしまいましょう。

画面から出てしまっても，一度視界に入ったものはサーチできることにします。サーチが切れるのは，より優先順位の高いサーチ物が視界に入ったときか，一定時間（5秒以上）画面外の敵に対し，サーチしなかった場合としましょう。

これだけの機能を搭載してしまえば，あとは，3軸の割と自由な操作系でも，そこそこ操作可能と思います。

●パワーアップ！

操作方法などの話も出てきて，GALAXY-KNIGHTSも，ようやくゲームの企画らしい内容になってきました。では，もう少し突っ込んでゲーム性の話を続けましょう。

ゲーム作りで忘れてはならないことは，ゲームの目的です。およそ真っ当なゲームには，必ず目的があります。

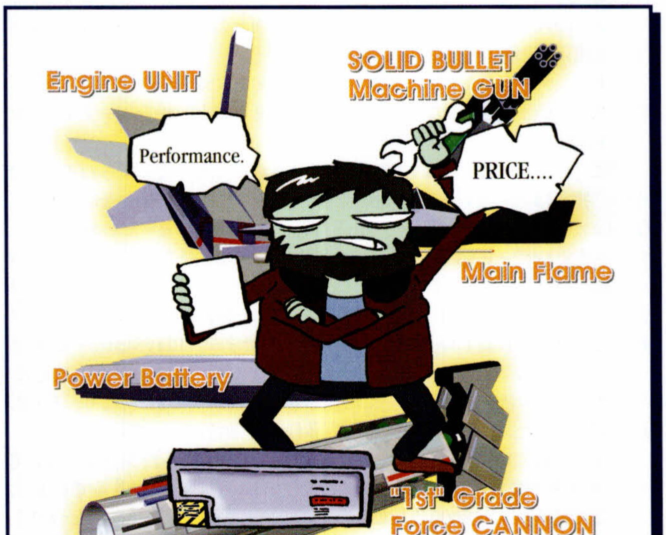
単なるシューティングゲームであれば，それは「敵の殲滅」だったり，「生き残って最終ステージまで行く」だったりします。RPGやアドベンチャーの場合，「目的地に行く」「敵を倒す」「宝を見つける」など，それぞれのミッションに従って，さまざまな目的が条件として設定されています。

GKの場合もご多分に漏れず，さまざまなミッションが設定でき，シナリオ次第でさまざまな展開が用意できるように作っていきます。そのなかで「宝を見つける」タイプのイベントが，本当にありがたみのあるものになるには，宝を手に入れた結果が，なんらかのかたちでプレイヤーに反映されるというシステムがいちばん簡単，かつ，効果的です。

それにいちばんよいのは，プレイヤーの機体の性能を決める「パーツ」や「アイテム」というかたちで，宝を用意することではないでしょうか。手に入れたパーツを利用して，機体をカスタマイズし，次のゲームでより有利に戦えるようにするわけです。

ただ，そうすると，初心者にはなんのアドバンテージもない……といったことになりかねないので（編注：上級ゲーマーほど楽になるゲーム性では意





味がない)，初心者は初心者としての「ハンデ」が得られるように考えてみましょう。

そこで，GALAXY-KNIGHTSでは，以下の形式で，「お宝」を用意することにします。まず，カスタマイズできる項目を設定します。

・エンジン

機体の運動性能を決めます。

・武器

攻撃能力，攻撃の威力を決めます。また，ビームとミサイルと爆弾と地雷では，ぜんぜん動作が違うので，そういう動作も決定します。

・シールド

機体の防御能力，反撃能力を決めます。

・アイテム

パーツのように性能が固定するのではなく，その場限りで特殊能力が使えるものとします。使うと，当然ながら消費されてなくなってしまいます。

次に，これをどう用意するかを決めましょう。

プレイヤーが勝手に作れるようだと，「お宝」の意味が薄れてしまいます。でも，できるものだったら，たとえば「ポストペット」の「おやつ」のように，参加者が自由に作れる側面もほしいですし……。

そうだ，いい手があります！

プレイヤーの機体に，「基本性能」を表す数値を設け，この数値が，戦果によって変わるようにしておきます。これに従って，取りつけられるパーツが決まるようにすれば，性能のいいパーツを取りつけたいときは，それなりの機体に「育てて」おく必要がある，という風になり，一方パーツを作る側は仕様にあってさえいれば，自由に作れる，ということで，八方丸く収まるではないですか。

育った機体に「貧弱な」装備だと，初心者よりも性能が落ちるようにしておけば，初心者に対するハンデも可能になります。

あとは，パーツの価値と，性能の対比を決める関数を用意するだけです。ただ，「だけ」といっても，プレイした感じが反映されないといけないものですし，きちんとした仕様を固めるには，実験と試行錯誤がある程度必要になります。実際の式は，動くものを組み上げていく次回に回し，今回は骨組みを作っておきましょう。

まず，各パーツは「グレード」というランク分けをすることにします。グレードが等しいものは「対等な」価値があるものとします。グレード数が多いほど，ゲームで登場するパーツを増やすことができますが，その分ゲームのバランスが難しくなります。

そこで，グレードは16段階として，各グレードを微調整するためさらに各グレードを「レベル」と呼ぶ段階に分けましょう。このレベルは微調整用なので，0～255という細かい値を与えようと思います。

あと，特別なステータスを表すために，条件ビットなどを設けようと思います。これら条件を含め，グレード数などもまとめると，だいたい以下のようなデータフォーマットになります。

label 「物体名」
　パーツ/アイテムの名前を指定します（全角16文字）
icon 「アイコン」
　対応するアイコンデータを示します
looks3D 「3Dデータ」
　対応する外見を示します
adapt 「装着時表示」
　アイテムを装着したことが，自機の外見に反映されるか，どこにどう反映するかを指定します
consume 「消耗品フラグ」
　使うとなくなります
abrasion 「磨耗フラグ」
　使うと徐々に磨耗します。後述の「耐久性能値」に関連します
usevent 「消費時イベント」
　使うと発生するイベントを指定します。というか，これがないと「使う」ことができません
ability 「性能フラグ」
　パーツにどのような性能があるか決定します
needs[] 「必要物リンク」
　このパーツを動かすために必要なほかのパーツのリスト
levels[] 「性能値」
　性能値列。上記性能フラグの分だけ存在

それぞれのメンバーの実データ形式は，実装時である次回に回します。アイコン，3Dデータ，イベントは，各実データに対する相対ポインタとします。それぞれの実データ形式も，次回の記事で決定しましょう。

また，性能値フラグの内訳も次回に回させていただきます。なんだか次回次回で，がっかりしてしまったかもしれませんが，それだけ次回の楽しみが増えたということにしておいてください。それでも，今回これだけの内容を決めれば，いろんなことができるのです。

このデータ形式は，大変柔軟性があります。たとえば，ある物体を，「エンジン」＆「ワープ性能」＆「投棄すると爆弾として使える」と設定し，さらに，必要物リンクに「ダイリチュウム結晶」とすると，あら不思議，U.S.Sエンタープライズ号のワープコアのできあがりです。

また，消費イベントを「プログラム」としてしまうと，柔軟どころの騒ぎではなくなり，たとえば消費イベントで，エネルギーを通常備蓄量＋20％消費，および照準スコープを出してメッセージで「対ショック，対閃光防御」を表示，トリガーを引くと発射，発射時に機体を激しく揺らす」とかできたとしたら，それはまんまヤマトの波動砲です。実際のイベント言語はもうちょっと制限がキツイので，上記そのままは難しいかしれませんが……。

そういった大層な武器ではなくても，いろいろ作ることができるでしょう。さらに，武器の場合の発射時のアニメーションなどもデータに含められたら，どんな武器でも柔軟に開発できるでしょう。

●通信の方法は？

ネットプレイをさんざん話題にしてきましたが，実際のプレイはどんな感じになるのでしょうか？　やはり，ネットワークゲームで人が集う必要がありますし，またシナリオを解いていくゲームとなれば，実際のゲーム開始前に，それなりの準備やブリーフィングが必要ですから，ゲームをする人たちが集まる「場所」が必要になるでしょう。

さて，そこで登場するのが第1部でも取り上げた「ロビーサーバ」です。ただ，通常のロビーサーバを構築するには，かなりの手間を必要とするので，この記事の規模には不向きだと考えました。そこで，ちょっとだけ仕組みを簡単にしたサーバを作ってみようと思います。目標としては，「一般のプロバイダに加入している人が，比較的簡単に設置できるサーバ」にしたいと思っています。

一般のプロバイダに設置可能なプログラムというと，やはりPerlによるCGIということになるでしょう。それでも，相手先のIPを取得したり，それを伝えることはできるので，一応ロビーサーバとしての役割は果たすと思います。もっとも簡単な『ロビーサーバ』は，ホームページをアクセスした人のIPを取得して，プレイヤーに対して表示するものです。欠点としては，現在進行しているプレイの内容が把握できないため，

あるプレイのセッションがいまも続行中かどうかを判定できない
誰がどのセッションに接続しているかわからない
セッションが満員であるかどうかを判定できない

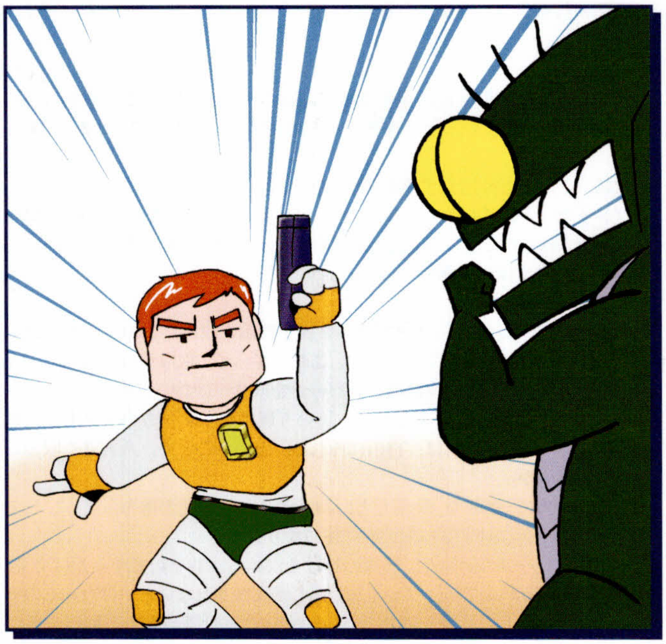
という点が挙げられます。
そこはもう，プレイヤーの自己申告に頼るしかないですが……。一応，誰かがアクセスしてきたとき，あまりにも時間が離れているデータは削除していくようにしますが，あくまで気休めであることを認識しておいてください。

●ソロプレイもできないと……

ネットワークプレイの方法ばかり書いてきましたが，はたしてネットワークゲームだけでいいのでしょうか？　ゲームの中身をよく知らないのにいきなりネットワークに繋ぐのは嫌だ。こう考える方がいても全然不思議ではなく，むしろそれはごく当たり前の反応ではないでしょうか。また，ネットワークに繋ぐのは現在，どうしてもある程度お金がかかる場合がほとんどですから，できたらお金をかけずにゲームをやりたいこともあるでしょう。
こういった意味から，ネットワークに繋がない，スタンドアローンの状態でのゲーム＝ソロプレイは，あると便利，というより，ある程度必要なもののようです。
ただ，ここで問題があります。ソロプレイでうーんと育てた機体で，いきなりとんでもなく強いキャラクターを，ネットワークに持ってこれるという風になると，みんな育てるだけ育てた機体でネットに繋いでしまいます。
それでいい場合もあるでしょうが，シナリオで「お宝」を出すなどの事情を考えると，やはりネットワークでのゲーム＝ネットプレイはネットプレイで育ててほしい気がします。
ソロプレイとネットプレイを分離する方法を考えてみましょう。
簡単で確実なのは，「ネットプレイのキャラクターはネット（サーバ）に置く」という方法ですが，それには最低限ロビーサーバへのアップロードができないと難しいでしょう。実際，HTTP＋CGIのロビーサーバでは，接続を自動化するだけでもかなり難しいですし，この案はこのレベルで没です。
もうひとつの方法としては，セーブデータを違うものにしてしまうという方法です。ただ，データに印をつけるだけだと，このゲームはソースリスト公開ですから簡単にデータを書き換えられてしまいます。
まあ，ある程度のパワーのある人だったらなんでもやれてしまうのでしょうが，ここはなるべくそういう「技」を極力抑える方法を考えましょう。すぐに思いつくのは，データ全体を暗号化し，暗号化の方法をネットプレイとソロプレイで異なる方法を採用するという手です。
さらに，ソロプレイとネットプレイでデータの内容そのものも変えてしまうという手もありますが，これはかなりの手間を要します。
データの内容を変えるのではなく，パッキングを変えるのならば，さして難しくないので，まずはこの手法で作ることにしておきましょう。あとは，簡単な暗号化を加えれば，普通には手を出せないものとなります。まあ，データに手を加えるような「兵（つわもの）」であれば，逆に不正もしないでしょうし，今回はこれでよしとしておきましょう。

●通信対戦のシナリオって？

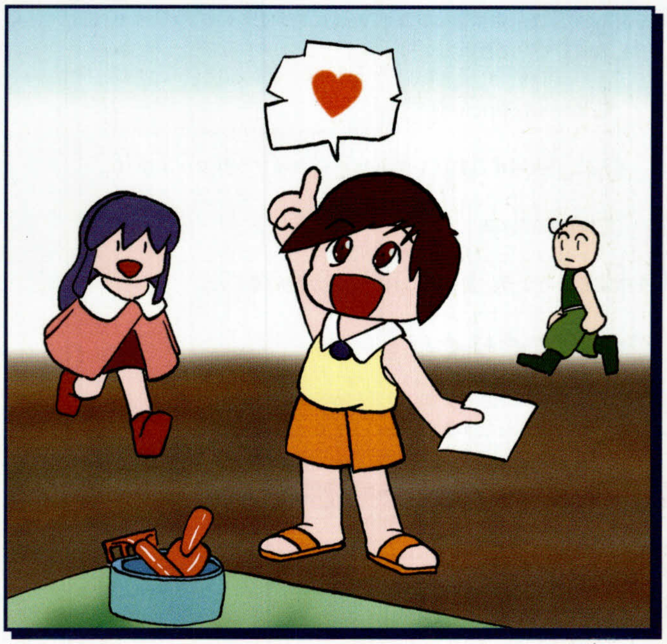
さて，ここまでシナリオシナリオと書いてきましたが，実はこれ，従来のゲームでいうシナリオとは，かなり異なるものを指しているのです。ソロプレイのゲームの場合は，プレイヤーは主人公です。それがどんな意味を持つかはさておき，ゲームはプレイヤーを中心にして進行します。
しかし，これがネットワークプレイの場合，複数人のプレイヤーがいるわけです。はて，誰を主人公にしたらよいのでしょうか？　みんなが主人公……と，いうわけにもいきませんね。しかし，ゲームをやっているプレイヤーにしてみれば，ソロプレイだろうが，ネットプレイだろうが，相変わらずゲームでの主役は自分であり，自分を中心にゲームが回ってほしいのは，当然といえば当然でしょう。
結果は，まるで「主人公様ご一行」といったありさまになってしまいます。
正直これでは，ゲームのシナリオは成り立ちそうにありません。本来お話を作るという場合，登場する人物のリアクションに対して，物事が進行していくさまを書いていくことになりますが，ネットプレイの場合，この「劇中に登場する人物」を特定できないという辛さがあるのです。
こういったシナリオを作る方法を考えてみましょう。とりあえず，２通りのやり方が考えられます。
ひとつ目は，あらかじめ複数人の人物が活躍する話を作っておき，参加したプレイヤーを，人物に割り当てていき，全部の人物に配役が収まった時点で，そのシナリオへの参加を締め切るというものです。この場合，プレイヤーの参加人数が少ない場合は，その人物はシナリオ内にあらかじめセットしたダミーに差し替えられるようにします。
利点は，プレイヤーが配役を変わることで同じシナリオでも違った遊び方ができるという点，欠点は，それぞれの配役ごとのデータを用意しなければならず，また，ダミーの場合の対応も用意する必要があるので，データ作





成がかなり面倒だという点でしょうか。

2つ目は，「主人公とその他大勢」という話を作る方法です。最初にゲームを始めた人が主人公となり，ほかの人たちはその他大勢を演じることになります。いってみれば「ゲームするものこの指止まれ」という人が最初にいて，この人が「メイン」で，メインの人がきた人に配役を振ります。いわば，最初にきた人が「ゲームマスター」になって，ロールプレイングをするようなものです。



対戦なんだから，勝敗も必要だ

対戦の準備もできたし，シナリオをどう作るかも決めました。では残るのは「シナリオの成果」でしょう。

第2部の3章目で，「お宝」については書きましたが，それだけではいちいちものを決めなくてはならなくて面倒です。そこで，「お金」の概念を持ち込みましょう。これだったら，「お金を稼いで」ものを買ったり，逆に人に報酬を払ってものを頼む，という話の展開が可能になります。基本的に，お金は仕事の報酬か，あとは「持っているものを売る」ことで手に入るようにすればよいでしょう。使うのは基本的には情報も含め，「なにかを買うとき」というのは，現実のお金と同じです。

ただ，「銀行」や「金融」がないため，お金自体を使ってなにかをすることはできません。ゲームのシステムに「ベースとなる街」を作れるように拡張していけば，こういった金融業や，お店を「開業」することも可能になるでしょうが，それはまた別の話ということにしましょう。

先ほどから「シナリオ」と連呼していますが，シナリオで大切なのは始まりと展開，そして「終わり」です。ゲームでいうシナリオの終わりは「終了条件」ということになります。これは，基本的には「お話が成就すればよい」というのがスタンスですが，「なにをどうすればお話が成就するか」は決めておいたほうがよいでしょう。これは，

敵の殲滅
チームに分かれて戦い，ある条件（時間や，やられた数など）が整ったときの勝敗の状態
ある地点に先に到達した
ある相手を倒した

などと決めることができます。

上記の条件だけで，たいていの試合は実現できます。これら条件，および前述のお金，店，アイテム，すべてを単一の「イベント言語」にて対応する予定です。イベント言語は残念ながら次回の紹介としますが，この機構の公開によって，さまざまなシナリオが登場するのを期待したいですね。

第3部　とにかく動くものがほしい

やっと本題！

ここまで薀蓄を長々と読んでいただきましたが，いよいよお待ちかねのプログラムに突入します。プログラムを作るからにはターゲットとなる環境を特定する必要があります。本来なら，DirectX7が動く環境ならなんでもOK！　とかいいたいのですが，限定しないと雑誌記事の範囲に収まらないため，まとめさせていただきます。

●ソフト環境

開発言語：
Visual C++（Ver.6.0以上）
（開発にはVisualStudio Professional Editionを使用しています）

ボーランドC++Builderも，ターゲットで考えてはみたのですが，前準備や過程など，どのステージでも違いが多すぎるため，今回は見送らせていただきます。

DirectX7 SDK

当然必須ですね。CD-ROMには7.0a SDKが入っているのでできたらそちらを利用してください。

OS：Windows98
Windows95や，2000も対応チェックしたかったのですが，煩雑になるので割愛。

●ハード環境

さすがに千差万別でしょうから，いくつかテストケースを用意してみました。当社のマシンで，私が適当に使えるマシンをチョイスしています。本当は，P3マシンと噂のGeForce256も入れたかったのですが，それは編集部に任せましょう。（U）さん。よろしくお願いしますね（編注：確認済みです）。

Type-1
K6-2/400MHz
Creative Graphic Blaster Extream
FastEarther-2
メモリ256MB

Type-2
Celeron/400MHz
Creative Voodoo Banshee
Fast Earther-2
メモリ128MB

Type-3
IBM-Aptiva E58
(PentiumII/300MHz)
メモリ192MBに増設済み。

おまけ
VAIO PCG-505EX
(MMX Pentium/233MHz)
メモリ64MBに増設済み。

とまあ，以上のような構成で始めてみたいと思います。

始める前に……

さあ，早速プログラムを……といきたいところですが，まだプログラムすら組み始めていませんのでちょっと待ってください。

ここで，Visual C++で，DirectX7を使うのが初めて，という方もいらっしゃるでしょうし，なにごとにも手順というものがあります。

まずは環境のセットアップから参りましょう。さて，皆さんはすでにVisual C++ (またはVisualStudio) のセットアップを済まされているものとします。DirectX7をセットアップされていない方は，付録のCD-ROMを起動しDirectX7.0a SDKのsetupを実行してください。

このとき，以前のバージョンのDirectX SDKをインストールしている方は，アプリケーションの追加と削除を使って，以前のバージョンのSDKを削除してからインストールに進んでください。この際，セットアップされるディレクトリはデフォルトのC:¥mssdk だということでお話を進めます。

さて，DirectX7の導入も終わり，いよいよ始めましょう。

……おっと，その前にまだありました。Visual C++に，「DirectX7で開発するぞ！」と教えてやる必要があります。これは簡単で，まずVisual C++を起動し，

「ツール」→「オプション」→「ディレクトリ」タグ

と選んで，「表示するディレクトリ」を「インクルード ファイル」にして，以下を追加してください。

C:¥mssdk¥include

さらに，「表示するディレクトリ」→「ライブラリファイル」に，

C:¥mssdk¥lib

を指定しましょう。さあ，これで準備が整いました。

3Dでぐりぐり

さて，いよいよ動くプログラムにかかりましょう。ただ，ここでまた少しだけ前置きをいただきます。この上まだなにかあるのか？　と，うんざりさ

リスト1

```
// ohx4.cpp
//

/*------------------- ヘッダーファイル＆マクロ定義 ------------------------*/

#define STRICT


#include "stdafx.h"
#define EXT
#include "ohx4.h"

char application_name[] = "OhX-4 cyberhead's sample";

// このコード モジュールに含まれる関数の前宣言:
ATOM            MyRegisterClass( HINSTANCE hInstance );
BOOL            InitInstance( HINSTANCE, int );
LRESULT CALLBACK  main_window_proc( HWND, UINT, UINT, LONG );
LRESULT CALLBACK  About( HWND, UINT, WPARAM, LPARAM );
LRESULT CALLBACK IPcon( HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam );

/*------------------- 外部変数 ------------------------------------------*/



/*------------------- 本体コーディング ---------------------------------------*/

/*
   メインウインドウを取得して、属性を与える
*/
bool   init_window( HINSTANCE hInst,int cmdshow )
{
WNDCLASS wndclass;
hInstApp = hInst;

// ウインドウ・クラス作成（詳細はＷｉｎｄｏｗｓＡＰＩに関する各資料を参照してください）
   wndclass.style          = CS_BYTEALIGNCLIENT|CS_HREDRAW|CS_VREDRAW;   // ウイン
ドウのスタイル定義（サイズ変更時の再描画、ＢＹＴＥアライメント調整）
   wndclass.lpfnWndProc    = (WNDPROC)main_window_proc;
       // メインウインドウのイベント処理プロシジャ
   wndclass.cbClsExtra     = 0;
   wndclass.cbWndExtra     = 0;
   wndclass.hInstance      = hInst;
   wndclass.hIcon          = NULL;
   wndclass.hCursor        = LoadCursor(NULL,IDC_ARROW);
   wndclass.hbrBackground = (HBRUSH)GetStockObject(BLACK_BRUSH);
   wndclass.lpszMenuName   = MAKEINTRESOURCE(IDC_OHX4);
   wndclass.lpszClassName = application_name;
// ウインドウ・クラスの登録
   if( !RegisterClass( &wndclass ) ){ return false; }
// メインウインドウの確保
   hwndApp = CreateWindowEx(
       WS_EX_CLIENTEDGE,
       application_name,
       application_name,
       WS_OVERLAPPEDWINDOW,
       CW_USEDEFAULT,CW_USEDEFAULT,640,480,
       (HWND)NULL,
       (HMENU)NULL,
       hInst,
       (LPSTR)NULL );

   ShowWindow( hwndApp, cmdshow );
   UpdateWindow( hwndApp );
   return true;
}

/*------------------- ウィンドウ処理 ----------------------------------*/
LRESULT CALLBACK main_window_proc(HWND handle,UINT msg,UINT window_param,LONG
param2 )
{
   switch(msg){
   case WM_ACTIVATE:
       switch( window_param ){
       case WA_ACTIVE:
           restart_draw();
           break;
       case WA_CLICKACTIVE:
       case WA_INACTIVE:
           pause_draw();
           break;
       }
       break;
   case WM_ACTIVATEAPP:   // アプリケーションの状態に合わせて動作を制御する
       if ((BOOL)window_param ) restart_draw(); else pause_draw();
       break;
   case WM_KEYDOWN:
       switch( window_param ){
       case VK_ESCAPE:
           DestroyWindow( handle );
           break;
       default:
           break;
       }
       break;
   case WM_DESTROY:
       PostQuitMessage(0);
```

れている方もいるでしょうが，もう少しだけおつきあいしてください。

VC++など，最近の統合環境を備えた言語では，おしなべて「プロジェクトの功罪」というのがあります。まあ，昔のmakefileと大差ないといえばいえるのですが，違うのは「存在が見えにくい」ということです。そして，DirectXの場合，必要な設定がこのプロジェクトの中に含まれてしまうため，ちょっとした問題が起きてしまうのです。

たとえば，今回のサンプルのように，雑誌についているプログラムでは，ワークスペースやプロジェクトが同梱されているため，大して考えなくてもソースを導入してコンパイルすれば動かすことができます。しかし，いざ自分で新たにプロジェクトを起こして，DirectXアプリケーションを作ろうとする場合，前章のオプション設定だけでは上手くコンパイルできません。

そして，ソースとしてすぐ閲覧できるところにその答えがないため，

「なんでサンプルは動くのに，俺の作ったプログラムは動かんの？」

となりがちです。

DirectXは，一般に新規作成→プロジェクト→Win32 Application を選択し，

「単純なWin32アプリケーション」

を選んで制作に入ります（もちろん，Hello - Worldや空のプロジェクトからだって始められますが，こっちが簡単です）。

でも，ここで作ったプログラムで，DirectDrawなどの機能を呼び出す，たとえばいきなり，

DirectDrawCreateEx（～）;

とかやってもエラーになってコンパイルできません。ライブラリを特定できないからです。そこで，

「プロジェクト」→「設定」→「リンク」

で，実際に使うライブラリを列記してやる必要があります。たとえば，オブジェクト/ライブラリモジュールに，

ddraw.lib d3dim.lib d3d d3drm.lib d3dx.lib d3dxof.lib
dxguid.lib dinput.lib dplayx.lib dsound.lib

と追加しておけばOKです（ここまで書かなくてよい場合が多いですが…）。

これでコンパイルすれば，無事にコンパイルできます。これまでのサンプルや紹介記事で，ちょっと気になったので，触れておきました。

さて，紆余曲折ありましたが，いよいよです。

では，手始めに3Dでぐるぐる動く画像を作ってみましょう。モデリングデータを使うことも考えましたが，時間的な余裕から，今回は見送らせていただき，簡単な生成物を使って画面を動かしていきます。

さらに，DirectX7の中で，今回新たに用意された「ヘルパー」ライブラリである，D3DXユーティリティライブラリを使い，かなりお手軽に作っていこうと思います。

```
        break;
    case WM_ENTERMENULOOP:      // メニューが選択されたら動作を停止する
        pause_draw();
        break;
    case WM_COMMAND:            // 選択されたメニューを実行
        switch( window_param ){

        case IDM_EXIT:
            DestroyWindow( handle );
            break;

        case IDM_ABOUT:
            DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, handle, (DLGPROC)About);
            break;

        }
        break;
    case WM_EXITMENULOOP:       // メニューの選択が解除されたら動作を再開する
        restart_draw();
        break;
    default:
        return DefWindowProc( handle,msg,window_param,param2 );
    }
    return 0L;
}

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow )
{
MSG msg;
HACCEL hAccelTable;

    // アプリケーションの初期化を行います:
    if( !init_window( hInstance, nCmdShow ) ) { return false; }
    hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_OHX4);
    if( !init_input() ){ return false; }
    if( !init_D3DX() ){ return false; }
    if( !start_D3DX() ){ return false; }

    init_surface();//  テクスチャ初期化
    init_shapes(); //  形状初期化
    init_objects();//  ３Ｄマップ初期化
    init_player(); //  プレイヤーステータス初期化

// メイン メッセージ ループ:
    msg.wParam = 0;
    while( 1 ){
        if (PeekMessage(&msg,0,0,0,PM_REMOVE)) {
            if (msg.message == WM_QUIT)
                break;
                TranslateMessage(&msg);
                DispatchMessage(&msg);
            } else {
                if( !realtime_render() ) PostQuitMessage(0);
                player_drive();
            }
        }
    release_objects();
    release_shapes();
    release_input();
    release_surface(); // サーフェイスを開放する
    release_D3DX();    // Direct3DXの終了処理
    end_application(); // アプリケーションの終了処理
    return msg.wParam;
}

// バージョン情報ボックス用メッセージ ハンドラ
LRESULT CALLBACK About( HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam )
{
    switch( message )
    {
        case WM_INITDIALOG:
                return TRUE;

        case WM_COMMAND:
            if( LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL )
            {
                EndDialog(hDlg, LOWORD(wParam));
                return TRUE;
            }
            break;
    }
     return FALSE;
}
// バージョン情報ボックス用メッセージ ハンドラ
LRESULT CALLBACK IPcon( HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam )
{
    switch( message )
    {
        case WM_INITDIALOG:
                return TRUE;

        case WM_COMMAND:
            if( LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL )
            {
                EndDialog(hDlg, LOWORD(wParam));
                return TRUE;
            }
            break;
    }
     return FALSE;
}
```

なお，以下のプログラムではWin32 APIや，その他のDirectXの機能も，特に断りなく使っていきます。

今回のサンプルでは，まず生成物体を出して，それを操作できるようにしてみようと思います。モデリングした物体は，次回の実際のゲームで扱います。

さて，DirectX7関係の書籍も最近ようやくぽちぽちと出始めていて，目を通したことのある方もおられると思いますが，多くの書籍がVisualBasic対応を大きく取り上げているのに対し，C++環境での変化についてはあまり取り上げられていません。これは実に悲しいことです。なぜなら，遥かに「人に優しくなった」DirectX7の側面を見せているのは，今回紹介するD3DXヘルパーライブラリの存在だと思うからなのです。

D3DXライブラリは，これまでいちばん面倒で複雑だったDirectDrawとDirect3Dの「初期化」と，2Dと3Dの混在処理を，いとも簡単に済ませることができるようにしてくれます。

流れ的には，

```
        // ウィンドウコンテキスト取得

    // ここらへんはWin32 APIを参照

    WND hwndApp;
    // アプリケーションのウィンドウハンドル

    WNDCLASS wndclass; // ウィンドウクラス登録用の構造体

if ( !RegisterClass ( &wndclass ) ){ return false; }

hwndApp = CreateWindowEx ( ~ );

    // D3DXライブラリ初期化

HRESULT   hr = D3DXInitialize ();

// これだけ

    // デバイスの列挙，ビデオモードの列挙
    // 必要な場合行う，必要でなければ，次のD 3D Xオブジェクト
      取得時に，
    // 所定の数値にD3DX_DEFAULTを指定すればよい
    // D3DXコンテキスト取得と，環境設定

LPD3DXCONTEXT   lpD3DX;

HRESULT hr = D3DXCreateContextEx ( ~,&lpD3DX );

// この1関数だけで，いままで面倒くさかった数々の初期化ができて
しまいます！

    // DirectDraw/Direct3DDevice/Direct3Dオブジェクトの取得
```

リスト2

```
/*
   Ｏｈ！Ｘ４号
   ＧａｌａｘｙＫｎｉｇｈｔｓサンプル
   Ｄ３Ｄ Ｘライブラリ手続き部分
*/
#include   "stdafx.h"
#include   "ohx4.h"

/*
   D3DXライブラリを初期化
*/
bool   init_D3DX()
{
HRESULT hr;

// Direct3DXの初期化
   hr = D3DXInitialize();
   if( FAILED(hr) ) { return false; }
   return true;
}

// Ｄ３Ｄ Ｘデバイスコンテキスト取得
bool CreateD3DXContext()
{
HRESULT hr;
   hr = D3DXCreateContextEx(D3DX_DEFAULT,          // デバイス。今回はウインドウモードなので
デフォルト
           0,                                      // スクリーンモード。今回はウインドウモー
ドなので0
           hwndApp,                                // ウインドウ
           NULL,                                   // ウインドウ･ハンドル
           D3DX_DEFAULT,                           // ビット深度（色）
           D3DX_DEFAULT,                           // ビット深度（アルファ・チャネル）
           D3DX_DEFAULT,                           // ビット深度（深度バッファ）
           D3DX_DEFAULT,                           // ビット深度（ステンシル・バッファ）
           D3DX_DEFAULT,                           // バックバッファ数
           640,                                    // 幅
           480,                                    // 高さ
           D3DX_DEFAULT,                           // リフレッシュレート
           &lpD3DX);

   if( FAILED(hr) ) { return false; }
   return true;
}

// 光源初期化
void   init_light()
{
D3DLIGHT7 lg;

   ZeroMemory(&lg, sizeof(D3DLIGHT7));

   lg.dltType = D3DLIGHT_POINT;

   lg.dcvDiffuse.r = 1.0f;
   lg.dcvDiffuse.g = 1.0f;
   lg.dcvDiffuse.b = 1.0f;

   lg.dcvAmbient.r = 0.3f;
   lg.dcvAmbient.g = 1.0f;
   lg.dcvAmbient.b = 2.0f;

   lg.dcvSpecular.r = 1.0f;
   lg.dcvSpecular.g = 1.0f;
   lg.dcvSpecular.b = 1.0f;

   lg.dvPosition.x = 1.0f;
   lg.dvPosition.y = 9.0f;
   lg.dvPosition.z = 1.0f;

   lg.dvAttenuation0 = 1.0f;
   lg.dvRange = D3DLIGHT_RANGE_MAX;

   lpD3DD->LightEnable(0, TRUE);
   lpD3DD->SetLight(0, &lg );
}

/*
   Direct3DX
   動作環境をセットアップする
*/
bool start_D3DX(void)
{
HRESULT hr;
   CreateD3DXContext();

   // サポートしているサーフェイス・フォーマットを取得する
   surface_format = D3DXGetMaxSurfaceFormats(
       D3DX_DEFAULT,
       NULL,
       D3DX_SC_COLORRENDERTGT);

   surface_format_list = new D3DX_SURFACEFORMAT[surface_format];

   for (DWORD i=0; i<surface_format; ++i) {
       D3DXGetSurfaceFormat(
           D3DX_DEFAULT,
           NULL,
```

LPDIRECTDRAW7
lpDD;
lpDD = lpD3DX->GetDD();

LPDIRECT3D7
lpD3D;
lpD3D = lpD3DX->GetD3D();

LPDIRECT3DDEVICE7 lpD3DD;
lpD3DD= lpD3DX->GetD3DDevice();

// D3Dデバイスオブジェクトによるデバイス設定。

lpD3D->SetRenderState(~);

// サーフェイス確保，初期化（必要に応じて）

LPDIRECTDRAWSURFACE7 lpBackBuffer; lpBackBuffer = lpD3DX->GetBackBuffer(0);

LPDIRECTDRAWSURFACE7 lpPrimary; lpPrimary = lpD3DX->GetPrimary();

// メインループ

// 画面クリア

HRESULT hr = lpD3DX->Clear(D3DCLEAR_TARGET [| D3DCLEAR_ZBUFFER]);

//
描画コマンド発行

HRESULT hr = lpD3DD->BeginScene();

if(SUCCESS(hr)){

// 必要に応じ，D3DXヘルパー，DirectDraw，Direct3Dなどの描画命令を列記。

lpD3DD->EndScene();

}

// 画面更新（フリップ，ブリット等は自動判断される）

lpD3DX->UpdateFrame(0);

// 終了までメインループへ

```
        D3DX_SC_COLORRENDERTGT,
        i,
        &surface_format_list[ i ] );
   }

   // 初期化したDirectDraw,Direct3Dオブジェクトを取得
   lpD3D  = lpD3DX->GetD3D();          if( lpD3D == NULL ) return false;
   lpD3DD = lpD3DX->GetD3DDevice();    if( lpD3DD== NULL ) return false;
   lpDD   = lpD3DX->GetDD();           if( lpDD  == NULL ) return false;

   hr = lpD3DD->SetRenderState(D3DRENDERSTATE_DITHERENABLE, TRUE);
   // ディザ
   hr = lpD3DX->SetClearColor(D3DRGBA(0.1f,0.4f,1.0,0));
        // 描画面をクリアするときの色を設定

   init_light();

   // マテリアル設定
   D3DMATERIAL7 material;
   ZeroMemory(&material, sizeof(D3DMATERIAL7));
   material.diffuse.r = 1.0f;
   material.diffuse.g = 1.0f;
   material.diffuse.b = 1.0f;
   material.specular.r = 1.0f;
   material.specular.g = 1.0f;
   material.specular.b = 1.0f;
   material.power = 3.0f;
   lpD3DD->SetMaterial(&material);

   return true;
}

/*
   サーフェイス初期化,
   今回はまだテクスチャを使わないので、
   関数はダミー。
*/
bool init_surface(void)
{
HRESULT hr = S_OK;
   return true;
}

/*
   サーフェイスの開放処理
*/
bool release_surface(void)
{
   xRelease(texp);
   return true;
}

/*
   Direct3DXの終了処理
*/
bool release_D3DX(void)
{
   xRelease(lpDD);
   xRelease(lpD3D);
   xRelease(lpD3DD);
   xRelease(lpD3DX);
   xDelete( surface_format_list );
   return true;
}

/*
   アプリケーション終了処理
*/
bool end_application(void)
{
HRESULT hr;

   hr = D3DXUninitialize();
   if (FAILED(hr)) { return false; }
   return true;
}

/*
   スクリーンモード変更時の処理
*/
bool restore_surface()
{
HRESULT hr;
int     ct;

   hr = DDERR_NOEXCLUSIVEMODE;
   ct=0;
   while( hr == DDERR_NOEXCLUSIVEMODE && ct<12 ){
       if( ct>0 ){ Sleep(500);    } /* リトライの場合、0.5秒待つ */
       ct++;
       hr = lpDD->TestCooperativeLevel();
   }
   if( SUCCEEDED( hr ) ) {
       hr = lpDD->RestoreAllSurfaces();
       if ( SUCCEEDED( hr ) ) {// サーフェイスの内容を復元
           if( init_surface() ){ return true; } // 正常に終了
       }
   }

   // エラー発生
   return false;
}
```

```
    // サーフェイスその他のリソース解放

  lpPrimary->Release();

    // オブジェクト解放
  lpD3DD->Release();
  lpD3D->Release();
  lpDD->Release();
  lpD3DX->Release();

    // D3DXライブラリ終了処理
  HRESULT hr = D3DXUninitialize();

    // アプリケーション終了
```

となります。

各リザルトステータスは，それぞれチェックして（if (FAILED (hr)) {～}などとします），失敗時の処理を入れることをお忘れなく。

従来のDirectX7と比べ，初期化が，

```
  D3DXInitialize();
  D3DXCreateContextEx();
```

の2関数だけですべて終わっていることに注目してください（デバイスの列挙をするともっと長くなりますが……）。

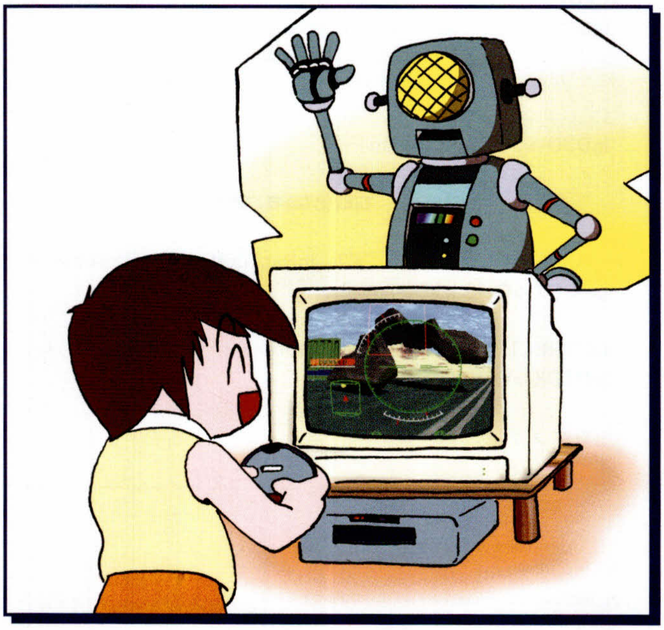
しかも，ここですべてのオブジェクトの初期化まで終わっているため，あとは，D3DXヘルパーライブラリに対して，たとえば，

```
  lpD3DX->Get****();
```

というメッセージを与えてオブジェクトを確保すれば，準備が整ってしまいます。

わずかこれだけで，3D描画システムが構築できてしまうのですから，D3DXヘルパーライブラリは，なかなか恐ろしいものがあります。

もっとも，処理をポカしたときは恐ろしく，特にオブジェクトの解放を間違うと，たちまち「不正な処理を実行しました」となった挙句，システムを巻き込んで死にます。こうなったら潔く再起動しましょう。まあ，ここらへ



リスト3

```
/*
   ＯＨ！Ｘ４号
   ＧａｌａｘｙＫｎｉｇｈｔｓサンプル
   制御入力部分
*/

/* ヘッダーインクルード   */

#include   "stdafx.h"
#include   "ohx4.h"
#include   <dinput.h>

LPDIRECTINPUT7          lpDI=NULL; //  DirectInput7オブジェクト
LPDIRECTINPUTDEVICE7  lpDID=NULL; //  DirectInput7デバイスオブジェクト

// Keyboard からの入力にDirectInput を初期化

bool   init_input()
{
HRESULT hr;
DIPROPDWORD dw;
// ＤＩＲＥＣＴＩＮＰＵＴ７オブジェクト確保
   hr = DirectInputCreateEx(hInstApp,DIRECTINPUT_VERSION,IID_IDirectInput7,(void
**)&lpDI,NULL );  if( FAILED(hr) ) return false;

// ＤｉｒｅｃｔＸ７デバイスオブジェクトを，キーボードに対して用意
   hr  =  lpDI->CreateDeviceEx(  GUID_SysKeyboard,IID_IDirectInputDevice7,(void
**)&lpDID,NULL ); if( FAILED(hr) ) return false;
   hr = lpDID->SetDataFormat(&c_dfDIKeyboard ); if( FAILED(hr) ) return false;
   hr  =  lpDID->SetCooperativeLevel(  hwndApp,DISCL_NONEXCLUSIVE  |  DISCL_FORE
GROUND); if( FAILED(hr) ) return false;

// バッファ・サイズを設定
   dw.diph.dwSize = sizeof(DIPROPDWORD);
   dw.diph.dwHeaderSize = sizeof(DIPROPHEADER);
   dw.diph.dwObj = 0;
   dw.diph.dwHow = DIPH_DEVICE;
   dw.dwData = sizeof(DIDEVICEOBJECTDATA) * 100;
   lpDID->SetProperty(DIPROP_BUFFERSIZE, (LPCDIPROPHEADER)&dw);

// 入力制御開始
   lpDID->Acquire();

   return true;
}

// 入力キーコードから，ゲームの内部処理コードを生成
// 何でわざわざそうするかというと，ＤｉｒｅｃｔＩｎｐｕｔを隠蔽するためと，
// 入力装置がキーボード以外になってもここで差を吸収し，
// 互換性を保つ事が出来るようにするため。

WORD   GetKeys()
{
BYTE   keys[256];
HRESULT    hr;
WORD   ret;
   ret = 0;
   hr = lpDID->GetDeviceState(256,keys);
   if( hr == DIERR_INPUTLOST ){
       lpDID->Acquire();
   } else {
       if( keys[DIK_SPACE      ] & 0x80 ){ ret |= PAD_SHOOT;   }
       if( keys[DIK_LSHIFT     ] & 0x80 ){ ret |= PAD_FORWARD; }
       if( keys[DIK_LCONTROL   ] & 0x80 ){ ret |= PAD_CMD;     }
       if( keys[DIK_Z          ] & 0x80 ){ ret |= PAD_LOCK;    }
       if( keys[DIK_UPARROW    ] & 0x80 ){ ret |= PAD_UP;      }
       if( keys[DIK_LEFTARROW  ] & 0x80 ){ ret |= PAD_LEFT;    }
       if( keys[DIK_RIGHTARROW] & 0x80 ){ ret |= PAD_RIGHT;   }
       if( keys[DIK_DOWN       ] & 0x80 ){ ret |= PAD_DOWN;    }
   }
   return ret;
}

// ＤｉｒｅｃｔＩｎｐｕｔ終了処理
void   release_input()
{
   xRelease( lpDID );
   xRelease( lpDI );
}
```

んはC++のプログラム開発ではよくあることですが……。

さて，今回のサンプルでは，このD3DXで，3Dの簡単な街の上をぐりぐり動き回るものを作ってみます。また，今回特にデータは用意せず，D3DXライブラリ内蔵の図形描画を利用してみました。もちろん，次回にあわせて拡張できるように，データを使った描画も考えていますが，データのコンバートプログラムなど，重たいプログラムがいくつもあるため，見送りました。生成物には，なぜかよく3Dサンプルで見かける「ティーポット」も交じっていましたが，さすがにこれも遠慮しました。

今回は，動作を見やすくするためと，メニューを表示したかったので，あえてウィンドウベースのプログラムとしました。でも，これをフルスクリーン版に改造するのは，D3DXライブラリのおかげでとっても簡単です。

```
D3DXCreateContextEx();
```

のなかで，2番目の引数(flags)に，

```
D3DX_CONTEXT_FULLSCREEN
```

を指定し，5番目の引数をNullではなく，4番目の引数と同じウィンドウハンドルに変え，10番目，11番目の引数を，

```
D3DX_DEFAULT
```

とするだけです。ね？　なんて簡単なんでしょう。

ソースの内訳ですが，ウィンドウコンテキストの処理と，メインループは，ohx4.cにまとめました。D3DX周りの初期化，終了処理は，d3dx.cにまとめてあります。参考にしてください(ヘッダはこれだけd3dxdrive.hとなっています)。

また，今回説明を割愛しましたが，DirectInputについて，キーボードの処理を作成しています。

```
control.c
```

です。コードも短いですし，詳細はコメント，およびSDKドキュメントを参照してください。

実際の描画関連は，render.cにまとめてあります。……といっても，実体は数行で(~->Draw(); だけ，という話もある)，ほとんどはカメラのマトリクス計算に費やしています。

では実際に動かしてみましょう。

```
¥Cybrhead¥Release¥ohx4.exe をダブルクリック
```

(IE 5でシングルクリックを設定している人はシングルクリックですが，こんな細かいツッコミはなしにしましょう)して，起動してください。

起動後，ウィンドウに収まった街並み(？)画面が出てくれば万歳です。

リスト4

```
/*
   Ｏｈ！Ｘ４号
   ＧａｌａｘｙＫｎｉｇｈｔｓサンプル
   レンダリング部分
*/

/* ヘッダーインクルード   */

#include   "stdafx.h"
#include   "ohx4.h"

/*---------------------------------------------
   アプリケーションの動作を一時停止する
---------------------------------------------*/
void pause_draw()
{
   if (lpDD) lpDD->FlipToGDISurface();          // GDIサーフェイスを表示する
   DrawMenuBar( hwndApp );                      // ウインドウのメニューバーを描画する
   RedrawWindow(hwndApp, NULL, NULL, RDW_FRAME); // ウインドウのフレームを描画する
// while (ShowCursor(TRUE)<0) {};               // マウスカーソルを表示する
}

/*---------------------------------------------
   停止していたアプリケーションの動作を再開する
---------------------------------------------*/
void restart_draw()
{
// while (ShowCursor(FALSE)>=0) {};             // マウスカーソルを消す
}

/*
   実処理
*/
bool realtime_render(void)
{
HRESULT hr;
OBJ3D  *obp;
DWORD  type;
D3DXMATRIX matRotate,matTranslate,mat,matX,matY,matZ,matView;
D3DXVECTOR3    vec;
   D3DXMatrixPerspectiveLH( &mat,1.0f,1.0f,1.0f,128.0f );

// カメラマトリックスを求める
   D3DXMatrixRotationX( &matX,camera_rud.x );
   D3DXMatrixRotationY( &matY,camera_rud.y );
   D3DXMatrixRotationZ( &matZ,camera_rud.z );
   matView = matY * matX * matZ;
   lpD3DD->SetTransform( D3DTRANSFORMSTATE_PROJECTION, mat);
// Ｚバッファ＆バックバッファ・クリア
   lpD3DX->Clear(D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER);
// 描画の開始
   hr = lpD3DD ->BeginScene();
   if( SUCCEEDED( hr ) ){
       obp = objtop;
       while( obp !=NULL ){

       //  ローカル回転マトリックスを求める
           D3DXMatrixRotationX( &matX,-obp->rudder.x );
           D3DXMatrixRotationY( &matY, obp->rudder.y+D3DXToRadian(180) );
           D3DXMatrixRotationZ( &matZ,-obp->rudder.z );
           matRotate = matZ * matX * matY;

       //  中心座標をグローバル変換
           vec = obp->pos - camera_pos;
           D3DXVec3TransformNormal( &vec,&vec, &matView );

       // 移動
           D3DXMatrixTranslation(&matTranslate,
               vec.x,
               vec.y,
               vec.z );

       // 変換行列を計算
           mat = matRotate * matView * matTranslate;
           lpD3DD->SetTransform( D3DTRANSFORMSTATE_WORLD, mat);
           type = shapelist[obp->type].type;
           if( type == FILE_DATA ){ //         ＬＷＯフォーマット(予定)の場合
               // lwo file data type is under construction !
           } else { // 単純図形の場合
               shapelist[obp->type].simplept->Draw();        //          実際の描画
はこの１行だけ！
           }
           obp = obp->next;
       }
       lpD3DD->EndScene();
   }

   // フレーム更新、フリップだろうがブリットだろうがこれでＯＫ
   hr = lpD3DX->UpdateFrame( 0 );
   if (hr == DDERR_SURFACELOST || hr == DDERR_SURFACEBUSY)
   {
       // サーフェイスが失われている場合の処理
       return restore_surface();
   }

   return SUCCEEDED(hr);
}
```

リスト5

```
/*
   Ｏｈ！Ｘ４号
   ＧａｌａｘｙＫｎｉｇｈｔｓサンプル２
   Copyright(c)2000,Cyberhead,Inc.
   DirectPlayドライブモジュール
*/

#include   "stdafx.h"
#include   "ohx4_2.h"
#include   <dplay.h>

LPDIRECTPLAY4 lpDP;
HANDLE        DPhandle;                // DirectPlay で受信に用いるイベント・オブジェクト
のハンドル
HANDLE        ReceiverHandle;          // メッセージ受信スレッドのハンドル
DWORD         ReceiverID;              // メッセージ受信スレッドのスレッドID
bool          newdataf = false;        // 新規データ、または同期データを送るフラグ
int               lost_count = 0;      // パケットロストに対応するため、同期データを送るため
のカウンタ

char   NAMESTR[]="OhX4_GalaxyKnights";              // アプリケーション名
char   PLAYERNAME[]="GK_Player";                    // プレイヤー名

#define    PLAYERS_LOOKS   4                        // プレイヤー外見（今回はトーラス,mapc
trl.cppの物体リストを参照）

DWORD WINAPI Receiver(LPVOID lpParams);
void   make_status_container( status_container *cp );

// {B01FFB24-C957-11D3-827A-0000F460DD22}
static const GUID guid_data =
{ 0xB01FFB24, 0xC957, 0x11D3, { 0x82, 0x7A, 0x00, 0x00, 0xF4, 0x60, 0xDD, 0x22 }
};

// DirectPlayオブジェクトを作成する
bool   init_directplay()
{
HRESULT    hr;
   CoInitialize(NULL);
   hr = CoCreateInstance(
        CLSID_DirectPlay,
        NULL,
        CLSCTX_INPROC_SERVER,
        IID_IDirectPlay4A,
        (LPVOID *)&lpDP );
   if( FAILED(hr) ){ return false; }
   DPhandle = CreateEvent(NULL, FALSE, FALSE, NULL);
   if (DPhandle == NULL){
        lpDP->Release();
        return false;
   }
   ReceiverHandle = CreateThread( NULL, 0, Receiver, NULL, 0, &ReceiverID );

   if ( ReceiverHandle == NULL ){
        lpDP->Release();
        CloseHandle( DPhandle );
        return false;
   }
   return true;
}

// 新しいプレイヤーを登録する
DPPlayer   *new_player( DPID player_ID,DWORD name )
{
DPPlayer *dpp;
   dpp = new DPPlayer;
   dpp->back = NULL;
   dpp->next = player_top;
   if( player_top!=NULL ) player_top->back = dpp;
   player_top = dpp;
   if( name != 0 ){ dpp->objp = make_object( name ); } else { dpp->objp = NULL;
}
   dpp->dplayID = player_ID;
   return dpp;
}

// プレイヤーを一人抹消
void   kill_player( DPPlayer *dpp )
{
   if( dpp==NULL ) return;
   if( player_top == dpp ) player_top = player_top->next;
   if( dpp->back!=NULL ) dpp->back->next = dpp->next;
   if( dpp->next!=NULL ) dpp->next->back = dpp->back;
   if( dpp->objp != NULL )    delete_object( dpp->objp );
   xDelete( dpp );
}

// DirectPlayオブジェクトの解放
// プレイヤーも全て消去する
void   release_directplay()
{
DPPlayer   *dpp,*ndp;
   xRelease( lpDP );
   CoUninitialize();
   dpp = player_top;
   while( dpp != NULL ){
        ndp = dpp->next;
        kill_player( dpp );
        dpp = ndp;
   }
   player_top = NULL;
}

// プレイヤーリストからプレイヤーを特定する
DPPlayer   *scan_players( DPIDplayer_ID )
{
DPPlayer   *dpp = player_top;
   while( dpp != NULL ){
        if( dpp->dplayID == player_ID ) return dpp;
        dpp = dpp->next;
   }
   return dpp; //  見つからなかった＝NULL
}

// キー操作パケットの送信
void   send_player_control( DPID id,DWORD key )
{
HRESULT    hr;
DPMSG_GENERIC      msgpack;
status_container   scon;
   if( multiplayer == FALSE ) return;
   if( newdataf==true ){  // 自分の現在の状態を既に参加しているメンバーに伝える
        newdataf = false;
        make_status_container( &scon );
        hr = lpDP->Send( id, DPID_ALLPLAYERS, 0, &scon, sizeof(status_container)
);
   } else {
        lost_count--;
        if( lost_count<=0 ){   //  パケットロストに対応するため、１２０サイクルに１回は、デー
タを丸投げする
             lost_count = 120;
             newdataf = true;
        }
        msgpack.dwType = ( key<<16 )| dpmsg_data_packet;
        hr = lpDP->Send( id, DPID_ALLPLAYERS, 0, &msgpack, sizeof(DPMSG_GENERIC));
   }
}

// 自分自身を生成する（開始直後はシングルプレイヤー）
void   make_myself()
{
        myself = new_player( 0,0 );
        myself->objp = make_object( 1 );
        multiplayer = false;
}

// サービス・プロバイダーの列挙

BOOL CALLBACK collect_probiders_callback(LPCGUID guid, LPVOID lpcp, DWORD size,
LPCDPNAME name, DWORD flag, LPVOID context )
{
HWND hwnd = (HWND)context;
LRESULT lr;
LPVOID lpbuf;

   lr = SendDlgItemMessage(
        hwnd,
        IDC_COMBOSP,
        CB_ADDSTRING,
        0, (LPARAM)name->lpszShortName );

   if ( lr == CB_ERR ){ return TRUE; }

   // サービス・プロバイダの情報を格納するメモリを確保する
   lpbuf = GlobalAlloc( GPTR, size );
   if ( lpbuf == NULL ){ return TRUE; }

   // サービス・プロバイダの情報をコピーし，コンボ・ボックスのアイテム・データとして割り当てる
   memcpy( lpbuf, lpcp, size );
   SendDlgItemMessage(
        hwnd,
        IDC_COMBOSP,
        CB_SETITEMDATA,
        (WPARAM)lr,
        (LPARAM)lpbuf );

   return TRUE;
}

// プロバイダリストの解放
```

```
void release_probiderlist(HWND hwnd, int items)
{
int max, i;
LPVOID lp;

   max = SendDlgItemMessage(
       hwnd,
       items,
       CB_GETCOUNT,
       0, 0 );
   for (i = 0; i < max; i++){
       lp = (LPVOID)SendDlgItemMessage(
           hwnd,
           items,
           CB_GETITEMDATA,
           i, 0);
       GlobalFree(lp);
   }
   SendDlgItemMessage(
       hwnd,
       items,
       CB_RESETCONTENT,
       0, 0);
}

// セッションの列挙

BOOL CALLBACK collect_sessions(LPCDPSESSIONDESC2 sessions, LPDWORD timeout,
DWORD flag, LPVOID context)
{
HWND hwnd = (HWND)context;
LPGUID guid;
LONG pt;

   if ( flag & DPESC_TIMEDOUT) { return FALSE; }

// 列挙で得たセッションリストをリストボックスに溜める
   pt = SendDlgItemMessage(
       hwnd,
       IDC_SESSIONS,
       LB_ADDSTRING,
       0,
       (LPARAM)sessions->lpszSessionName );

   if ( pt == CB_ERR ) return TRUE;

// セッションのガイドデータベースを作成、格納
   guid = (LPGUID)GlobalAlloc(GPTR, sizeof(GUID));
   if ( guid == NULL )return TRUE;
   *guid = sessions->guidInstance;
   SendDlgItemMessage(
       hwnd,
       IDC_SESSIONS,
       LB_SETITEMDATA,
       (WPARAM)pt,
       (LPARAM)guid );

   return TRUE;
}

// セッションリスト解放
void release_sessions(HWND hwnd, int item)
{
int         max, i;
LPVOID lpt;

   max = SendDlgItemMessage(
       hwnd,
       item,
       LB_GETCOUNT,
       0, 0);
   for( i = 0; i <max; i++ ){
       lpt = (LPVOID)SendDlgItemMessage( hwnd, item, LB_GETITEMDATA, i, 0 );
       GlobalFree( lpt );
   }
   SendDlgItemMessage(
       hwnd,
       item,
       LB_RESETCONTENT,
       0, 0);
}

// セッション選択ダイアログウィンドウ
BOOL CALLBACK session_dialog_proc(HWND handle, UINT Msg, WPARAM wParam, LPARAM
lParam)
{
HRESULT             hr;
DPSESSIONDESC2      sdesc;
int                 sno;
int                 retval = TRUE;

   switch( Msg ){
   case WM_INITDIALOG:// セッション一覧作成
       memset(&sdesc, 0, sizeof(DPSESSIONDESC2));
       sdesc.dwSize = sizeof(DPSESSIONDESC2);
       sdesc.guidApplication = guid_data;

       hr = lpDP->EnumSessions(&sdesc, 0, collect_sessions, handle, DPENUMSES
       SIONS_AVAILABLE);
       if (FAILED(hr)){
           MessageBox(handle, "セッション一覧が作れません", "コネクト失敗", MB_OK |
           MB_ICONWARNING);
           release_sessions(handle, IDC_SESSIONS);
           EndDialog(handle, FALSE);
       }
       return TRUE;
   case WM_COMMAND:   // セッション参加or新規作成
       switch (wParam) {
       case IDOK: //  OKボタンが押された=選択したセッションを受け取る
           sno =  SendDlgItemMessage(handle, IDC_SESSIONS, LB_GETCURSEL, 0, 0);
           memset(&sdesc, 0, sizeof(DPSESSIONDESC2));
           if ( sno != LB_ERR ){ // セッション起動
               GUID *guidsession;
               sdesc.dwSize = sizeof(DPSESSIONDESC2);
               guidsession = (GUID *)SendDlgItemMessage(handle, IDC_SESSIONS,
               LB_GETITEMDATA, sno , 0);
               sdesc.guidInstance = *guidsession;

               // 既存のセッションに接続
               hr = lpDP->Open(&sdesc, DPOPEN_JOIN);
               if ( FAILED(hr) ) retval = FALSE;
           } else { // OKボタンを押したとき，セッション未選択=新しいセッションを作る
               sdesc.dwSize = sizeof(DPSESSIONDESC2);
               sdesc.dwFlags = DPSESSION_MIGRATEHOST | DPSESSION_KEEPALIVE;
               sdesc.guidApplication = guid_data;
               sdesc.dwMaxPlayers = MAXPLAYERS;
               sdesc.lpszSessionNameA = NAMESTR;

               // セッションの新規作成
               hr = lpDP->Open(&sdesc, DPOPEN_CREATE);
               if( FAILED(hr) ) retval = FALSE;
           }
           release_sessions( handle, IDC_SESSIONS );
           EndDialog( handle, retval );
           return TRUE;
       case IDCANCEL:
           release_sessions(handle, IDC_SESSIONS );
           EndDialog(handle, FALSE);
           return TRUE;
       default:
           return FALSE;
       }
   default:
       return FALSE;
   }
   return TRUE;
}

// サービスプロバイダーリスト選択用ダイアログウィンドウ
BOOL CALLBACK probiderlist_dialog_proc(HWND handle, UINT Msg, WPARAM wParam,
LPARAM lParam)
{
   HRESULT hr;
   int iSelectedProvider;
   LPVOID lpConnection;

   switch( Msg ){
   case WM_INITDIALOG:// サービスプロバイダーリスト作成
       release_probiderlist(handle, IDC_COMBOSP);
       lpDP->EnumConnections(NULL, collect_probiders_callback, handle, 0);
       return TRUE;

   case WM_COMMAND:
       switch (wParam){
       case IDOK:
           // [OK]ボタンが押されたときの処理
           iSelectedProvider =SendDlgItemMessage(handle,  IDC_COMBOSP,CB_GETCUR
           SEL, 0, 0);
           if (iSelectedProvider != CB_ERR) {
               // サービス・プロバイダを初期化する
               lpConnection = (LPVOID)SendDlgItemMessage(handle, IDC_COMBOSP,
CB_GETITEMDATA, iSelectedProvider, 0);
               hr = lpDP->InitializeConnection(lpConnection, 0);
               release_probiderlist(handle, IDC_COMBOSP);
               if ( SUCCEEDED(hr) || (hr == DPERR_ALREADYINITIALIZED) ){
                   EndDialog(handle, TRUE);
               } else {
                   MessageBox(handle, "サービスプロバイダーの初期化に失敗しました",
                   NAMESTR, MB_OK | MB_ICONWARNING);
                   EndDialog(handle, FALSE);
               }
           } else {
               MessageBox(handle, "サービスプロバイダーを設定してください", NAMESTR,
MB_OK | MB_ICONWARNING);
```

```
          }
          return TRUE;
       case IDCANCEL:
          release_probiderlist(handle, IDC_COMBOSP);
          EndDialog(handle, FALSE);
          return TRUE;
       default:
          return FALSE;
       }
       break;
   default:
       return FALSE;
   }
   return TRUE;
}

// 対戦モードにする
void    connect_player( HWND hwnd )
{
HRESULT    hr;
DPID   dID;
// セッション開始
// マルチプレイヤーモードにする
   multiplayer = true;
   if (DialogBox(hInst, MAKEINTRESOURCE(IDD_CONNECT), hwnd,
(DLGPROC)probiderlist_dialog_proc)){
// セッション選択のダイアログ・ボックスを表示する
       if (DialogBox(hInst, MAKEINTRESOURCE(IDD_SESSION), hwnd,
(DLGPROC)session_dialog_proc)){
          // プレイヤーを作成する
          DPNAME dpName;

          // プレイヤーの名前を設定
          memset(&dpName, 0, sizeof(DPNAME));
          dpName.dwSize = sizeof(DPNAME);
          dpName.lpszShortNameA = PLAYERNAME;
          dpName.lpszLongNameA = NULL;

          // プレイヤーの作成
          hr = lpDP->CreatePlayer( &dID, &dpName, DPhandle, NULL, 0, 0);
          // 作成したプレイヤーのDPIDを保存
          myself->dplayID = dID;
          newdataf = true;

          InvalidateRect(hwnd, NULL, TRUE);
          UpdateWindow(hwnd);
       }
   }
}

// 回線切断する
// 自分以外のプレイヤーを消去
// セッションを終了
// シングルプレイヤーモードにする
void    disconnect_player( HWND hwnd )
{
DPPlayer   *dpp,*ndp;
   multiplayer = false;
   if (myself->dplayID != NULL)   {   // プレイヤーの削除
       lpDP->DestroyPlayer( myself->dplayID );
       myself->dplayID = NULL;
       dpp = player_top;
       while( dpp != NULL ){
          ndp = dpp->next;
          if( dpp != myself ) kill_player( dpp );
          dpp = ndp;
       }
       InvalidateRect(hwnd, NULL, TRUE);
       UpdateWindow(hwnd);
   }
   lpDP->Close();
}

// 既存プレイヤーを生成する
void    preset_player( DPPlayer *dpp,status_container *cp )
{
   dpp->objp->pos     = cp->position;
   dpp->objp->rotate  = cp->rotation;
   dpp->objp->rudder = cp->rudder;
   dpp->objp->move    = cp->speed;
   dpp->objp->type    = cp->name;
}

// 自分の情報をデータコンテナに詰め込む
void    make_status_container( status_container *cp )
{
   cp->dwType     = dpmsg_now_status;
   cp->name       = PLAYERS_LOOKS;
   cp->position = myself->objp->pos;
   cp->rotation = myself->objp->rotate;
   cp->rudder     = myself->objp->rudder;
   cp->speed      = myself->objp->move;
}

// メッセージ送受信のスレッドを用意
DWORD WINAPI Receiver(LPVOID lpParams)
{
   // メッセージを受信するスレッド
DPID   fromID,toID;
LPDPMSG_GENERIC msgbuf;
DWORD  msgbufsize;
HRESULT stat,hr;
HWND hwnd;
DPPlayer   *dpp;
status_container  scon;
   hwnd = (HWND)lpParams;
   while (WaitForSingleObject(DPhandle, INFINITE) == WAIT_OBJECT_0) {
       //メッセージの取り出し
       msgbuf = NULL;
       msgbufsize = 0;
       stat = DP_OK;
       while( SUCCEEDED( stat ) ){
          stat = lpDP->Receive(
                 &fromID,
                 &toID,
                 DPRECEIVE_ALL,
                 NULL,
                 &msgbufsize );                  // 受信バッファ作成
          if( stat != DPERR_BUFFERTOOSMALL ) break;
          if( msgbuf != NULL ) GlobalFree( msgbuf );
          if(  (  msgbuf  =  (LPDPMSG_GENERIC)GlobalAlloc(  GPTR,  msgbufsize  )
)==NULL ){
              stat = DPERR_OUTOFMEMORY;
              break;
          }
          // データを受信
          stat = lpDP->Receive(
                 &fromID,
                 &toID,
                 DPRECEIVE_ALL,
                 msgbuf,
                 &msgbufsize );
          if( FAILED(stat) ) break;

          // 受信したメッセージの処理
          switch( fromID ){
          case DPID_SYSMSG:  // システム・メッセージ
              switch ( msgbuf->dwType ) {
              case DPSYS_CREATEPLAYERORGROUP://ユーザーが作られた
                 //  新規参加者のために自分の状態を告げる
                 make_status_container( &scon );
                 hr  =  lpDP->Send(myself->dplayID,DPID_ALLPLAYERS,  0,  &scon,
sizeof(status_container));
                 break;
              case DPSYS_DESTROYPLAYERORGROUP://ユーザーが減った
                 dpp  =  scan_players(  ((LPDPMSG_DESTROYPLAYERORGROUP)msgbuf)-
>dpId );
                 kill_player( dpp );
                 break;
              }
              break;
          default:   //  各々のプレイヤーの挙動データ処理
              dpp = scan_players( fromID );
              if( dpp == NULL ){ //  相手は新規
                 dpp = new_player( fromID, PLAYERS_LOOKS );
                 dpp->objp->type = 0;           // 外見無しにしておく
              }
              if( msgbuf->dwType == dpmsg_now_status ){
                 preset_player( dpp,(status_container *)msgbuf );
              } else {
                 command_player( dpp->objp, *( (DWORD *)msgbuf ) );
              }
          }
       }
       if( msgbuf != NULL ) GlobalFree( msgbuf );
   }

   ExitThread(0);
   return 0;
}
```

カーソル左右で向きを変え，左SHIFTで前進できます。終了はESCキーです。また，カーソルの上下キーで，見上げたり，見下ろしたりできます。いろいろ試してみてください。

今回のプログラムは，わかりやすいようにかなり無駄な記述をしていますが，それでも，エントリーマシン程度の能力があればさくさく動かすことができるでしょう。よい時代になりましたねえ。

●繋いでみましょう

さて，簡単な街の表示ができたところで，次にこの町に友達を呼びましょう。そうです，今回の記事でさんざん訴えてきた「通信」を実現しようというわけです。使うのはもちろん，DirectXファミリーのなかで，通信を受け持つ DirectPlay です。

DirectPlayは，第1部でも述べたとおり「サービスプロバイダ」というかたちで，通信形態を自由に選べます。たとえば，高速なイーサネットで繋がっている場合と，モデムで繋がっている場合を，サービスプロバイダを取り換えるだけで，なんら意識することなくプログラムにできるのです。昔のプログラマは，「通信対応」などといおうものなら，1つひとつのプロトコルを死にそうな形相で締め切りに間にあわせて作っていたのですが，便利になったものですね。

また，DirectPlayでは，相手にデータが確実に伝わったことを保証する方式と，「垂れ流し」で，データを放送する形式などが，自由に選べるようになっています。今回のサンプルでは，割と高速に動くため，パケットを待つのがもったいないので，「垂れ流し」を採用してみました。

流すデータは，基本的には操作キーのデータで十分なはずなのですが，垂れ流しの場合，データを受け取り損なうと，お互いの位置が滅茶苦茶に狂ってしまいます（たとえば，Aの人は，自分ではビルの前に立っているつもりでも，Bの人のマシンの中では広場にぽつんと立っている，などということが起こりうるのである）。そこで，120サイクルに1回，ドカン，と，現在の座標や，移動値などを送りあうようにしました。

実際のところ，これと「同期式」データ通信とで，どちらがスムーズに動くのかは検証してみないとわかりませんが，とりあえずは相手がちゃんと動いて見えるので，今回はこれで置いておくことにします。最終的な通信プロトコルは，ゲームに仕立て上げる際に，再度検討しよう。

では，プログラムをざっと説明しましょう。

DirectPlayの本体もやはりオブジェクトです。ただし，DirectDrawなどと異なり，Windowsで標準的に用いられる「COMオブジェクト」というカテゴリのものでできています。したがって，DirectPlayオブジェクトを作成する際は，Windowsの標準手続きを使用してオブジェクトを作成する必要があります。

具体的には

```
CoInitialize(NULL);

// COMを初期化する
        hr = CoCreateInstance( CLSID_DirectPlay,～,&lpDP )
// DirectPlayオブジェクトを，インスタンスとして生成する。
```

という風になります。

ここで生成したDirectPlayオブジェクトを使い，イベントハンドルを作ります。DirectPlayの通信は，外部からの割り込みなので，イベントとして処理されるわけです。

割り込みである以上，いつ，どんなかたちでくるか予想がつかないため，メインのプロセスとは別に，常に監視する形のプログラムにしないと，メインのプロセスが待ちぼうけを食わされてしまい，プログラムの動作がガクガクになってしまう可能性があります。そこで監視用のプログラムは別途，スレッドを作り，その内部で通信を行うことにしましょう。当然，非同期でプレイヤーが参加してくることを想定する必要があるため，監視スレッドは，

「新しいプレイヤーが参加した」
「プレイヤーがいなくなった」
「データがきた」

という状態を処理するように作る必要があります。文章で説明すると非常に長くなるので，ここらへんはソースリストで確認してください。

dplaydrive.cpp の Receive () プロシージャが，監視スレッドの本体です。DirectPlayで通信を行う実体は，「プレイヤー」と呼ばれます。プレイヤーは固有のIDを持っていて，それでお互いを区別しています。DirectPlayでは，DPIDと呼ばれるデータがこれに相当します。

プレイヤーを生成するには，

```
hr = lpDP->CreatePlayer((DPID *)&dpID,～);
```

という具合に行います。これは，本プログラムでは，dplaydrive.cpp の connect_player () で行っています。

当然，データがきたときにはそのプレイヤーのIDが必要で，逆に送る場合も，自分が誰なのかを知らせるためにこのIDを送信データに付加しておきます。IDは，プレイヤー単位だけではなく，「グループ」単位でも，「全員」というカテゴリでも送受信が可能です。

DirectPlayそのものには関係しないんですが，シングルプレイヤーのルーチンと大きく異なり，

「ほかのプレイヤーと，マシンを直接操るプレイヤーの操作ルーチンは同じにしなくてはならない」

という点があります。このため，mapctrl.cppも，サンプル1からかなり手を加えています。参考にしていただけると幸いです。

実際の動作としては，

```
¥ohx4-2¥Release¥ohx4_2.exe
```

を実行してください（実際に接続するには，最低2台のPCが必要です）。起動後，「接続」の中の「接続」を押してください。

次に出てくるサービスプロバイダの欄は，欄の右端の▼を押して，適当なプロバイダを選択してください。接続相手を聞いてきますが，これは相手のIPアドレスか，わからなければ空欄で結構です。そうすると，セッション一覧（最初は空白）が出ますので，最初の接続ではそのままOKを押してください。2代目の接続をするとき，欄に，

```
OhX4_GalaxyKnights
```

というのが出ていれば成功です。これを選んで，OKを押してください。

しばらくすると，マップ上にドーナツが現れます。これがほかのプレイヤーです。いまのところ，マップをうろつくだけですが，まあ，相手がなにをしているか見るだけでも結構楽しめると思います。

＊　　＊　　＊

以上，バタバタと駆け足でここまで作ってきましたが，いかがでしたでしょうか。

「なんだこの程度か」

といわれてしまうとそれまでですが，今回のプログラムは，3D＋通信対戦＋入力の，かなり汎用なスケルトンになっています。いろいろいじって楽しんでもらえれば幸いです。

さて，次回は，これらのプログラムを利用して，シナリオなども絡め，いよいよゲームとして仕上げていくことになります。ご期待ください。

D3DRMで作る
3DシューティングゲームSION2000

菊地 功 Kikuchi Isawo

できるだけ手軽に，しかも典型的な3Dゲームということで，旧Oh!X読者の方々には懐かしのSION風シューティングゲームを作ってみました。昨今のDirect3Dを使えばRMでもこれくらいのことは簡単にできるのです。

最近のゲームの流行はポリゴンである。猫も杓子もポリゴンポリゴン，「単なるクオータービューじゃん」っていう古典RPGまで無駄にポリゴンを使う始末。ま，流行だからしょうがないか。そんなわけで，いまやゲームはポリゴンなしには語れないのである，っていうのは大げさだが，アメリカなんかではポリゴンじゃなきゃ売れないってのは事実のようである。Street Fighter EXあたりがそのあおりを食って作られたゲームらしい（編注：そうなのか？）。幸いというかなんというか，日本は伝統的なキャラクタゲームが根づいていて，まだまだそんなことはないが（もちろん日本のゲームメーカーが3Dのノウハウを持っていないというのもある），やはり3Dゲームというのは男のロマンである。

そんなわけで，ここではDirect3DのRetained Modeを使って，そこそこお手軽に3Dゲームを作ってみよう。ただし，一からすべてを解説することは不可能なので，本誌のDirect3D Retained Mode講座である程度の知識をつけた人を対象に，要所要所の説明を行うことにする。「マイガッ！なんてこった忘れちまったい」って人は，まずはバックナンバーを引っくり返してほしい。

旧Oh!Xの読者ならば，3DゲームといってSION（シオン）シリーズを思い出す人も多いだろう。山田氏制作による，3Dゲームの基本，奥スクロール（？）シューティングゲームである。その最終形態SION Ⅳは，横内氏のSLASHというポリゴンエンジンを利用して作られた，いま見てもかなりしっかりした3Dポリゴンゲームである（編注：ゲーム本体は作り込む時間が足りなかったのだけど……）。テクスチャがないのでディテールはそれなりだが。今回はこれを目標にしてみよう。コードネームはFakeSION，SION・のようなもの，である。

●まずは簡単な展望から

Direct3Dの場合，ユーザーのハードウェアによってパフォーマンスは極端に変わる。ローエンドからハイエンドまで何十倍もゲームの進行時間が違うというのでは，ゲームとして成り立たない。そこで，レンダリング時に前回からの経過時間を測定し，それに応じてオブジェクトを動かせば，少なくとも時間の流れは統一される。しかし，キー入力や当たり判定もそのタイミングで行えば，キーの反応の違いや，遅いマシンでは弾がすり抜けるなど，さまざまな弊害が起き，これまたゲームにならない。真面目にやろうと思えば，入力や演算などを行うターンは一定時間ごとに行い，レンダリング時はそれを補間してオブジェクトを配置することになるだろう。かなり面倒臭そうだ。

まあここまで説明しておいてなんだが，今回は時間がないこともあり，そんなことはせず，時間の流れはフレームレートに依存することにする。速い場合には適当にウェイトを入れて，もったいないがフレームレートを落とすということで，一定以上のパフォーマンスを持ったマシンならば制作側の意図どおりに遊べる，というようにしておこう。

視点はコクピットビュー。上下左右に動けるが，端はない。そうなると敵の出現ポイントが問題となるが，出現時点で自機の真正面を基準とする。敵は正面から現れ，適当に停滞したりもするが，撃墜されなければ後ろに逃げていく。この辺は2Dのシューティングゲームと同じようにしていいだろう。当たり判定も2Dとほぼ同様だ。2Dは矩形で判定するのに対し，3Dではボックスで行う。判定領域が細長かったり平たかったりすると，敵の基準軸に座標を変換する必要があるが，今回はそこまでは行わず，あくまでもワールド座標系の立方体を敵の判定領域とする。もう少し内部仕様を詰めると，カメラ位置は固定とし，自機の動きはそれ以外のオブジェクトの相対移動で表現する。そのほうがなにかと都合がいいのだ。いちいちすべてを動かすのは面倒だと考えるかもしれないが，どのみちほかのオブジェクトもすべて動いているわけだから，そのついでに相対移動量を加算してやればいいだけだ。

自機は，まっすぐ正面に向かって飛んでいく2連装のリニアカノンと，追尾ミサイルを装備する。SIONに倣って，リニアカノンはキーを押しっぱなしで撃ち続けることができるが，追尾ミサイルはボタンを押しておくことで敵をロックし，離すと発射するようにする。ついでに，1機を撃墜してから一定時間内に次を撃墜すればコンボとし，ヒットコンボで得点が上がるようにもしてみよう。

●コーディング

Direct3Dの初期化なんかについては，RM講座のほうで使っているEnumDeviceを含む適当なプログラムをスケルトンにする。とりあえずはオブジェクトの読み込みや配置は置いといて，背景の球面スクロールから。自機の動きにあわせて，背景の宇宙の画像をスクロールさせる。自機は上下左右に平行移動するだけで，向きは変わらないわけだから，正面の無限遠にある景色が動くわけがないという意見もあるが，そこはそれ，雰囲気である。十分遠い位置に十分大きなポリゴンを1枚立て，そこに書き割りとして背景テクスチャを貼るという手もあるが，いまいち気分がよくないので，DirectDrawで直接描くことにする（が，ひょっとしたらDirect3Dで描いたほうが速いのかもしれない）。背景のスクロールは，あんまりぐりんぐりん動かれても困るので，オフセット座標を管理する変数は実際の座標の10倍のレンジで取ってある。つまり，スクロールの最小単位は1/10ドットである。

背景画像はVGAサイズとした。これをバックバッファに対して，4回IDirectDrawSurface::Bit()する。もしゲームの解像度がこれよりも高い場合は拡大して，ちょうど画像がピッタリ画面にはまるサイズで描画される。

オブジェクトは，とりあえず自分の通常弾と追尾弾，それに敵と爆発を配置することにする。それぞれをIDirect3DRMFrame3に載せる必要があるのだが，当初これらの構造体を作りリンクアレイチェーン（コラム参照）にして，出現と消滅のたびに構造体とIDirect3DRMFrame3の構築と廃棄

をしていたところ，最初のうちはいいのだが，遊んでいるうちにどんどん重くなるという現象が起こった。確保と解放を繰り返していると，どうもメモリのフラグメンテーションが起こって，確保に時間がかかるようになるようだ。そこで，あらかじめ構造体の配列を用意することにした。

ただし，配列の数だけループを回していると，出現数が少ないときは時間の無駄だし，確保する配列の数でパフォーマンスが変わるというのも気持ち悪い。そこで，この配列の中でさらにアレイチェーンし，出現しているキャラクタの数しかループしないようにしている。制限数以内で，メモリ管理を自分でやっているようなものだ。また，IDirect3DRMFrame3のインスタンスも，必要になったときに初めてこの構造体内で確保し，そのキャラクタが消滅しても次に使い回すようにした。

通常弾は簡単だ。図1のような尾の長い三角錐をIDirect3DRMMeshBuilder3で作っておき，発射したときにこれを左右それぞれIDirect3DRMFrame3::AddVisual()してフレームに載せ，それをZ方向に動かすだけだ（Z軸回りに回転も加えている）。このとき，この弾は半透明にし，シェーディングしないようにしている。半透明にするには，

```
IDirect3DRMMeshBuilder3::SetColor()
```

でメッシュの色を設定するときに，引数としてD3DRGBA()マクロでα値を一緒に設定すればいい。

```
D3DRGBA(1.0,1.0,1.0,0.4)
```

と設定してあるが，これはRGBがそれぞれ1.0，つまり白で，αが0.4ということだ。αは小さいほど透明度が高く，0.0だと透明，1.0だと不透明となる。

シェーディングのほうも，IDirect3DRMMeshBuilder3::SetQuarity()メソッドでメッシュ単位に指定できる。引数として，シェーディングはフラット（D3DRMSHADE_FLAT），ライトはなし（D3DRMLIGHT_OFF），フィルモードはソリッド（D3DRMFILL_SOLID）を渡すが，このコンビネーションはD3DRMRENDER_UNLITFLATという定数でも定義されている。また，弾の尾の部分は外が表になるようにメッシュが作られているが，頭は内側を表にしてある。弾はまっすぐしか飛んでいかないので，見えるのは必ず後ろ姿であり，半透明の尾を透けて頭はより明るく（白く）見える（図2）。

追尾弾は，1発で10個のフレームを使っている。ひとつのフレームには幅0.1，長さ1.0のメッシュを載せ，それを連結させることで軌跡を表現する。先頭は不透明，後ろに行くにしたがって透明度を上げれば，綺麗な尾に見える。動かすときには，10個のフレームをすべて動かすのではなく，いちばん後ろを前に繋げて不透明にし，残りはαを1段階ずつ下げればよい。さて，追尾弾というからには，当然敵を追尾する。この方法を考えてみよう。いまベクトルABがあるとする。これを点Cに向けたいとしよう（図3）。ここで便利なのが外積である。高校で習ったはずだ。数式で表すと，

$$\begin{pmatrix} v.x \\ v.y \\ v.z \end{pmatrix} = \begin{pmatrix} v1.x \\ v1.y \\ v1.z \end{pmatrix} \times \begin{pmatrix} v2.x \\ v2.y \\ v2.z \end{pmatrix} = \begin{pmatrix} v1.y \times v2.z - v1.z \times v2.y \\ v1.z \times v2.x - v1.x \times v2.z \\ v1.x \times v2.y - v1.y \times v2.x \end{pmatrix}$$

となる。ここで，v1とv2は単位ベクトルである。すると，ベクトルvはベクトルv1およびv2に直行し，その長さはv1とv2の成す角θの正弦（sin）となる。これは，v1をベクトルvを軸にasin（|v|）回転させるとv2に一致することを意味する。追尾の話に戻してみよう。v1は弾の現在の進行方向，v2は弾の現在位置から敵までのベクトル（の単位ベクトル）に対応させると，外積によって回転軸と角度が得られることがわかる。Direct3DRMにはヘルパー関数として単位ベクトルを得る関数と，外積を得る関数，ベクトルの長さを得る関数が用意されているので，それらを使うと，

```
// v1は現在の弾の進行方向
D3DRMVectorNormalize( &v1 );
```

Column

●リンクアレイチェーン

変数や構造体を複数確保したい場合，配列を用いることが多いが，その数を動的に変化させたい場合は，次のような手法もよく取られる。

```
struct FOO {
  int someone;
  struct FOO ＊next;
}
```

nextメンバは，この次にくるFOO構造体のインスタンスアドレスを示している（最後ならばNULLを入れるのが一般的）。このような手法をリンクアレイチェーンという。数を増やしたければFOO構造体を新たに確保し，チェーンの最後のnextメンバにそのアドレスを格納する。あるいはチェーンの途中に入れたければ，その直前のnextメンバにアドレスを格納，新たに確保した構造体のnextメンバに次のアドレスを格納してチェーンを繋ぎ直すことで，配列のようにデータをずらすことなく，差し込むことができる。数を減らしたい場合も同様だ。また，nextメンバだけではチェーンを前から後ろへしかたどれないが，直前のアドレスを示すlastメンバも組み込めば，後ろから前へもたどれるようになる。このように，リンクアレイチェーンを用いれば配列のように初期確保数に制限されることや，無駄にメモリを浪費することなく，効率的に複数の変数・構造体などを管理することができる。

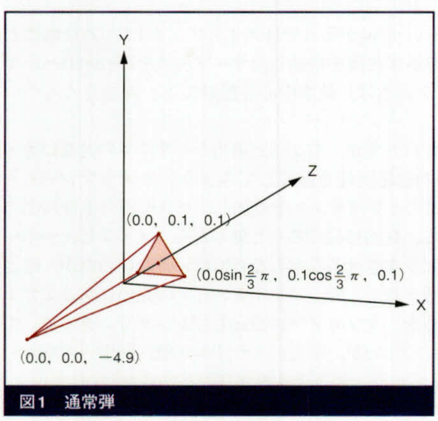


図1　通常弾


図2　リニアカノン

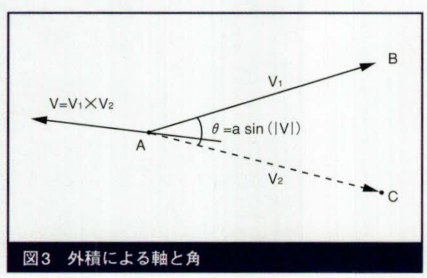


図3　外積による軸と角

```
// v2は現在の弾から敵までのベクトル
D3DRMVectorNormalize( &v2 );
// 外積
D3DRMVectorCrossProduct( &v, &v1, &v2 );
// v1とv2の成す角度
th = asin( D3DRMVectorModulus( &v ) );
```

となる。あとは弾が載っているフレームlpD3DRMFrmに対して,

```
lpD3DRMFrm->SetRotation( lpD3DRMScene, v.x, v.y, v.z, th
);
```

あるいはZ軸とY軸を制御して,

```
lpD3DRMFrm->GetOrientation( lpD3DRMScene, &v1, &v2 );
// Z軸の回転
D3DRMVectorRotate( &v1, &v1, &v, th );
// Y軸の回転
D3DRMVectorRotate( &v2, &v2, &v, th );
lpD3DRMFrm->SetOrientation( lpD3DRMScene, v1.x, v1.y,
v1.z, v2.x, v2.y, v2.z );
```

などとする(lpD3DRMSceneはシーンフレーム)。

前者のほうが圧倒的に簡単だが,進行方向を得るためにZ軸が必要な場合も多いので,私は後者の方法を取ったということにしておこう(実はいまのいままでなぜか上の方法を思いつかなかった)。

これだと間違いなく弾は敵に向かって飛んでいく。結果,敵がフィールドの外に逃げていなくならない限り,百発百中である。これはこれで面白くないので,ちょっと仕掛けをした。ひとつは一度に一定角度以上曲がれないというものだ。これは,thの最大値を設ければよい。

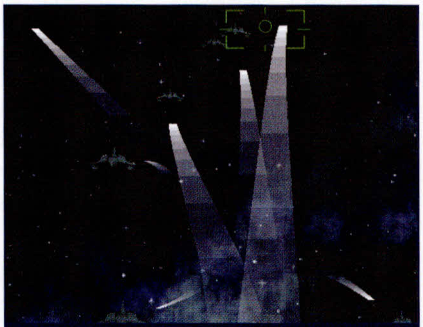


図4 追尾ミサイル

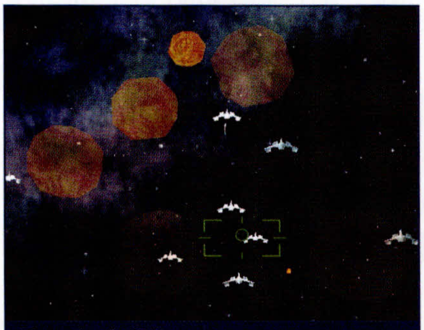


図5 ボンボリ爆発

```
if( th<-g_PI/24 ) th = -g_PI/24; else if( th>g_PI/24 ) th =
g_PI/24;
```

とでもすれば,それ以上は曲がらなくなる(g_PIはπ)。実際にはthは正の値しか取らないようなので,マイナスのときの判定は不要だ。もうひとつは,thが一定以上の角度だと,敵を見失ってしまうようにする。きっと追尾ミサイルは先端に取り付けられたカメラにより敵を認識しているのだろうから,そのカメラの画角から外れたら,ロストしてしまうのだろう。

敵は現時点ではダミーである。DirectX SDKについていたdropship.xを,ランダムに出現・動作させておこう。攻撃もまだしてこない。

爆発は,いわゆるボンボリ爆発である。球状のオブジェクトを作っておいて,それを徐々に大きくしていきながら,透明度を上げる。適当にテクスチャを貼っているが,弾同様シェーディングはしない。ただし,中にライトを1個仕込んである。爆発オブジェクト自体はシェーディングを行っていないので,このライトで照らされることはないが,爆発によって周囲の敵が照明される効果を狙ったものだ。点光源を埋め込むのがリアリティ的にはベターなのだが,重たいので平行点光源にした。この平行点光源は,光線のベクトルこそ光源とオブジェクトの座標から算出するが,あとは平行光源と同じ振る舞いをする。つまり,減衰もしないわけだ。まあ,点光源に比べれば圧倒的に軽いし,爆発ってのはきっと凄い明るさだろうから,見えている範囲では減衰しなくてもいいかってことで。

あとは細かい部分。たとえば追尾弾のロックとか。カメラは原点に固定してあるし,ロックできる範囲が画面の中央なので,特に難しい三次元計算は必要ないだろう。敵の座標が(x,y,z)ならば,x/zおよびy/zが一定の範囲内にあれば,そいつはロック範囲内にいると判断していい。ロックしたかどうかは敵の構造体が保持している。また,追尾弾を発射したあとは,追尾弾の構造体が追尾すべき敵の構造体のポインタを保持している。こうしておけば,直前に発射した追尾弾が敵に到達するなり消える前に,次々と同じ敵に対してロック・発射を行える。まあ,このあたりはプログラミング次第だ。ロックの表示に関しては,敵をサーチして,ロックされているものの座標を,

```
IDirect3DRMViewPort2::Transform()
```

でワールド座標からスクリーン座標に変換する。このメソッドではD3DRMVECTOR4D構造体のインスタンスが得られる。

```
typedef struct _D3DRMVECTOR4D {
    D3DVALUE x;
    D3DVALUE y;
    D3DVALUE z;
    D3DVALUE w;
}D3DRMVECTOR4D;
```

このメンバのうち,x/w,y/wがそれぞれディスプレイ上での2D座標となるので,そこにロックを示す画像を格納したサーフェスをDirectDrawで貼り付ける。これでロック表示は,敵が中心を離れても,画面をくっついて回るはずだ(図6)。

あと,最終的なゲームでは外すが,ちょっと適当なミサイルの先端にカメラをくっつけて,そいつの追尾映像を確認してみよう。カメラフレームとIDirect3DRMViewport2のインスタンスを追加し,ビューポートを画面の右下1/4の領域に設定する。真面目にやろうと思ったら,メインビューポート側にこの領域をクリッピングさせるなどしなくてはいけないのだが,確認だけなのでここでは深く考えない。で,アレイチェーンの先頭にあるミサイルのフレームの座標と向きを,サブカメラに設定してレンダリングする。ちょっと動きが速くてわかりづらいが,確実にミサイルが敵に向かって飛んでいることがわかる(図7)。

とりあえずここまでがサンプルgame1である(図8)。この時点ではまだ

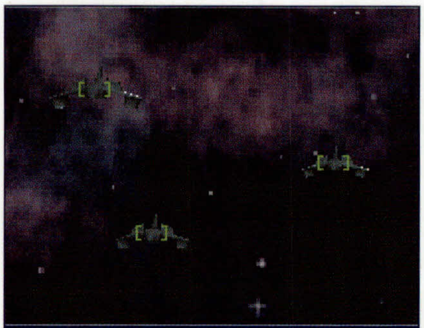
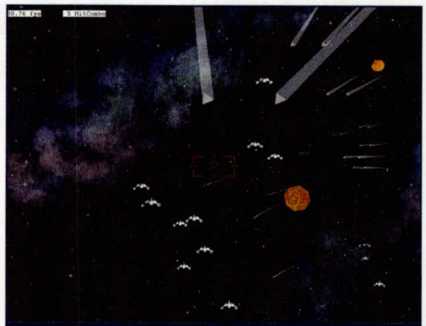

図6　ロックされたことを示すマーク


図7　ミサイル積載カメラの映像


図8　サンプルgame1

フレームレート調整のウェイトを入れていないので，速いマシンだとわけがわからないことになるかもしれない。HELを使うなどして遅くしてもらいたい。キー操作については，自機の動きがカーソルキー，通常弾がスペース，追尾弾はZキー押下でロック，離すと発射する。適当にいじって動作を確認してもらいたい。まあ，基本はこんなもんだ。

●敵のモーションをつける

当然敵も攻撃してこなくちゃいけないわけで，敵の種類によって飛行パターンや攻撃パターンは違うわけだ。そこで，まずはキャラクタのデザインをする。モデリングには六角大王を使ったが，Xファイルへのコンバートについては今号のRM講座のほうを見てもらいたい。で，完成したオブジェクトが表1である。相変わらずセンスないが，その辺はご容赦願おう。

まずはこいつらの動きを考える。Triangulmから。トライアングラムだかドライアンガルムだか発音はよーわからんが(名づけておいてそういうこというな)，こいつはザコ中のザコだ。まっすぐ飛んできて，適当なところまできたら，XY平面内の45度(あるいは135，225，315度)の軸を基準に向きを変える(図9)。どの軸を使うかは，自機を基準としたXY平面内で，敵のいる象限によって決まり，中心(つまり自分)に向くように敵の進路が回転される。たとえば45度の軸であれば，(1.0,1.0,0.0)のベクトルとなるので，それを基準に追尾弾のときと同じようにフレームを適当に回転させる。まあ，ほとんどなにも考えていないモーションだが，ザコだからこれでよしとしよう。

次はLepus。こいつもなんて発音するのかよくわからないが，やっぱりザコだ。Triangulmと違うのは，一定の位置でちょっと停滞，中心に向けてゆっくり浮遊しながら弾を撒き散らすこと。その後は同様に45度の軸を基準に進路を回転させるが，今度は自分から逃げる向きに軸を選択している。弾をばら撒くだけばら撒いたら，逃げていくというわけだ(図10)。

Vulpeculaにはもうちょっとまともな動きをつけよう。一定距離まで近づいたら，中心方向に旋回，そのままいったん向こうを向いて遠ざかる。その後，一定距離まで遠ざかったら，再び旋回してこちらへ向けて飛んでくる(図11)。1回目の旋回は，ほかのザコと同じで，単に旋回期間を長くするだけだ。それに対し，2回目の旋回は，正確にこちらに向けて方向転換している。ただし，このとき1パスで敵の向きと敵から自機までのベクトルによる外積で演算を行うと，外回りの回転をして画面から大きく外れてしまうことがある(図12)。そこで，いったん中心に向けて方向転換を行い，そこからさらに自機に向けて向きを変えるという2パスで軌道計算している(図13)。

ボスは置いといて，量産型のDemetriusにいこう。こいつはLepusとVulpeculaをあわせたような軌道を描く。適当な距離まで近づくと，停滞して弾を吐き，そのあとゆっくりと後退しながら方向転換，奥に向かって飛んでいき，また旋回してこちらへ飛んでくる。ただこいつがほかとちょっと違うのは，追尾ミサイルをかわす能力があるということだ。追尾弾は，追尾している敵に一定距離まで近づくと，その敵の構造体に対して警告フラグを立てるようにしてある。このフラグを得てDemetriusは，図14のように(敵のローカル座標の)Z軸に平行で，x=0.0,y=1.0を通る軸を中心に回転し，螺旋飛行でミサイルから逃れようとする。新座標の原点は旧座標で(sin(θ),1.0-cos(θ))で表されるので，フレームをそれだけ移動させ，さらにθの回転を加えればよい。ただし，移動座標はローカル座標系での話なので，IDirect3DRMFrame3::Transform()でワールド座標に変換してや

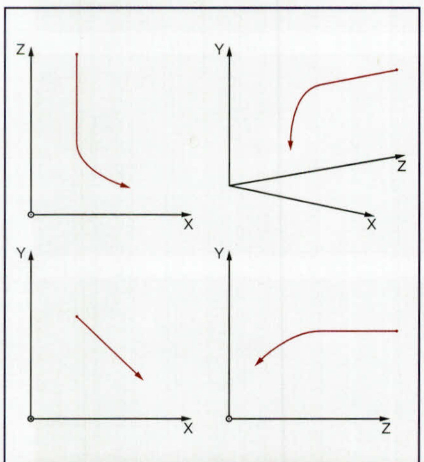


図9　Triangulmのモーション

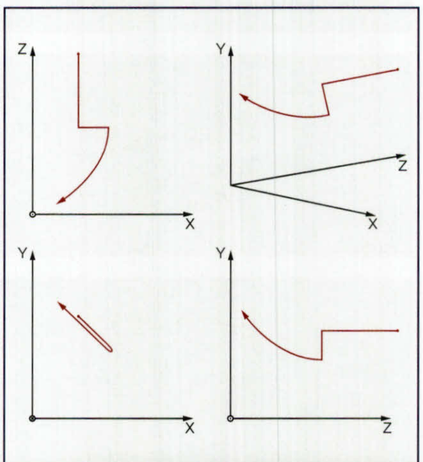


図10　Lepusのモーション

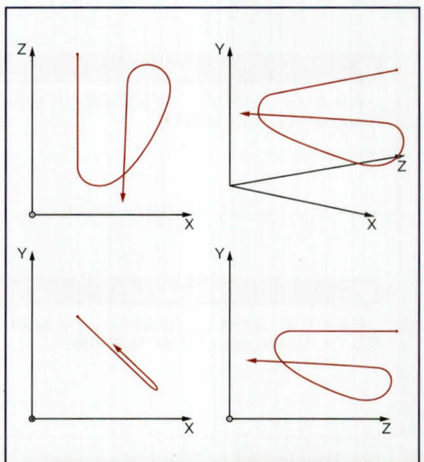
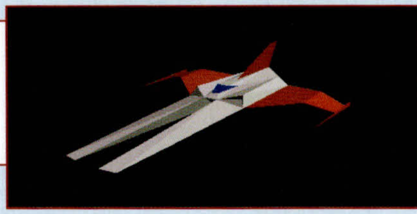
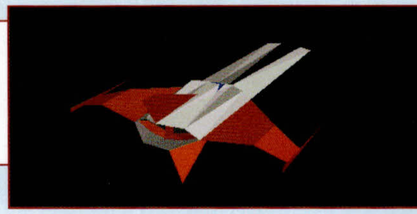


図11　Vulpeculaのモーション

表1 キャラクター一覧

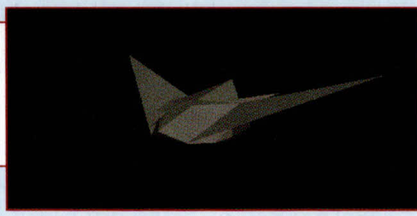
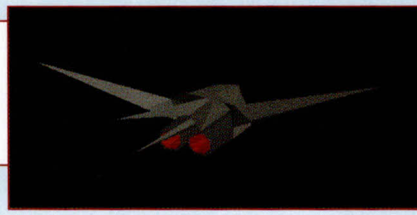
Astarte

こちらの機体。2連装リニアカノンと追尾ミサイルを装備する。Astarte（アスタルテ）は愛と豊穣の女神Isthar（イシュタル）のカナン地方での呼称。

Demetrius

いわゆるゲルググ（イメージはブラックサレナ）。敵のエースの少佐（または大佐）の乗る専用の機体。こちらとほぼ同等のシールドを持ち，追尾ミサイルの回避能力に優れる。ステージ1および2のボス。Demetriusとはクロアゲハの亜種。その形状はもとより，回避時の運動性からついた名。

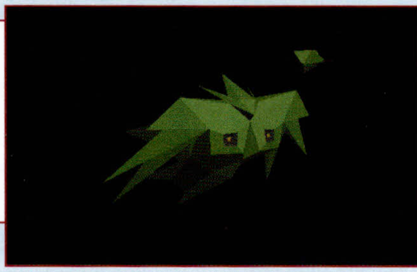
Demetrius mass-production type

かなり形が変わっているが，Demetriusの量産型。シールドは簡易になっているが，ミサイル回避能力は健在。Vulpecula同様，旋回して攻撃してくる。



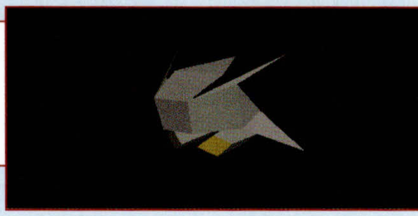
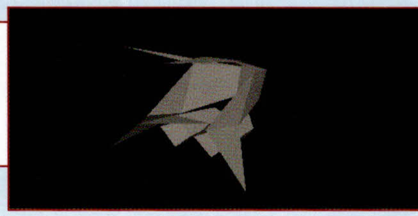

Eresh-ki-gal

ステージ3のボス。いわゆるビグザムかジオング（イメージはノイエ・ジール）。Demetriusを凌ぐシールド，およびミサイル回避性能を持つ。また，ファンネルを装備し，オールレンジ攻撃を仕掛けてくる。Eresh-ki-gal（エレシュ・キ・ガル）はメソポタミア神話の黄泉の国の女王で死の女神。



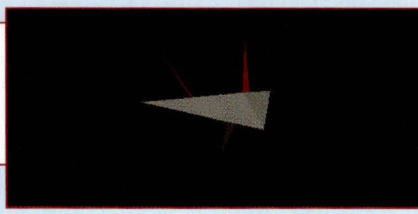
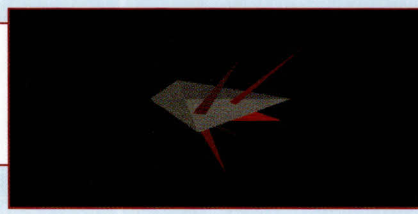

Lepus

いわゆるグフ。自機の前に停滞し，弾をばら撒いたあとはそれで逃げていく。Lepusはうさぎ座の意。

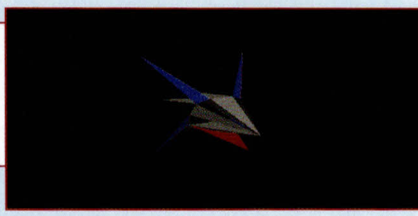
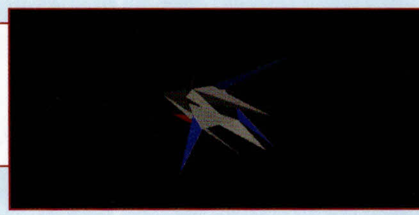
Triangulm

いわゆるザク，いやザコ。適当に自機の前を横切る。Triangulmはさんかく座の意。

Vulpecula

いわゆるドム。足が速く，目の前でいったん旋回して攻撃してくる。Vulpeculaはこぎつね座の意。

る必要がある。この運動は結構曲者で，割とミサイルを高い確率でかわしてくれる。Demetriusは1発では撃破できないようにしてあるので，それとあわせてゲーム中ではなかなか厄介なヤツだ。

敵の弾は3種類考えた。ひとつは単にその時の自機に向けて撃つ弾。こいつはなにも考える必要がない。２つ目は自機の最近の動きから着弾点を予測して，そこに向けて撃つ弾。「最近の動き」は，平均移動量ではなく，フレームごとに次のような式で算出している。

```
avespeed = avespeed * 0.8+speed * 0.2
```

avespeedは予測に使用する速度，speedはそのときの速度である（実際にはx,yそれぞれについて計算する）。最近の速度ほど影響が大きく，昔のものは徐々に薄められていく。このようにしておけば，真面目に配列を取って平均するよりも簡単だ。この速度を基準とすると，図15のような着弾予測ができる。この図において，lは敵と自機の距離，v1は先ほどの予測速度，v2は弾のベクトルである。ここで着弾予想時間をtとすると，次のような式が成り立つ。

```
v2^2 t^2 = l^2 + v1^2 t^2 - 2 l v1 t cos θ
```

これはtについての二次方程式であり，これを解いて時間がわかれば，着弾予想地点もわかる。そこに向けて弾を撃つわけだ。なお，θを求めるには，今度は内積を使う。数式で表すと，

$$\begin{pmatrix} v1.x \\ v1.y \\ v1.z \end{pmatrix} \cdot \begin{pmatrix} v2.x \\ v2.y \\ v2.z \end{pmatrix} = v1.x * v2.x + v1.y * v2.y + v1.z * v2.z$$

となる。この値は，ベクトルv1とv2の成す角θの余弦（cos）である。内積もヘルパー関数が用意されており，D3DRMVectorDotProduct ()という関数を使えば一発だ。

３種類目の弾は，追尾弾だ。演算の仕方はこちらの追尾ミサイルと同じだが，視界の外から側面に当てられるとゲームにならないので，一定よりも浅い角度にはならないようにしてある。また，ほかの種類の弾についても，視界外の敵は弾を撃たないようにしている（画角を60度としている）。

弾の形状は三角錐，これを適当に回転させる。こちらの弾同様，半透明でシェーディングなしだ。まっすぐに飛ぶ弾，つまり１，２番目の種類の弾は赤と白でフラッシュ，追尾弾は黄色と白でフラッシュさせる。まあ，適当に動き回っていれば通常弾が当たることはほとんどないので，黄色いのだけを意識してしっかりよけていれば，ダメージを受けることはあまりないだろう。

ここでちょっと入力デバイスを考えたい。いままではGetAsyncKeyState ()というWinAPI関数で処理してきたが，ジョイスティック（パッド）に対応させるのは必須だろう。そこで，DirectInputを使ってみる。DirectInputはデバイスとしてキーボードやマウスを選択することもできるので，キーボードで遊ぶ場合もこれを使う（こちらのほうが速いはずだが）。ソースコード自体はDirectX SDKに付属のGameInputサンプルから引っ張ってきたが，それほど難しいものではない。必要なインタフェイスはIDirectInput7とIDirectInputDevice7の2つ。

```
DirectInputCreateEx( hAppInst, DIRECTINPUT_VERSION,
IID_IDirectInput7, (LPVOID *) &pDI, NULL);
```

としてIDirectInput7を作成したら（pDIはIDirectInput7のポインタ），DirectDrawの初期化と同じように，

```
pDI->EnumDevices( DIDEVTYPE_JOYSTICK, EnumJoysticks
Callback, &guidJoystick, DIEDFL_ATTACHEDONLY );
```

としてデバイスを列挙，選択する（EnumJoysticksCallbackはコールバック関数，guidJoystickは返されるGUID）。コールバック関数では，たとえば列挙されたデバイスがハットを持っているかとか，フォースフィードバックに対応しているかなどでふるいにかけることができるが，このゲームでは上下左右とボタンが2つあればいいだけなので，特に機能チェックはしない。普通のジョイスティックならば，それくらいはついているだろう（いや，ハンドルとかならわからないか。昔はサターンのバタフライハンドルでバーチャファイターをやったもんだが）。ついで，

```
IDirectInput7::CreateDeviceEx(
    REFGUID rguid,
    REFIID riid,
    LPVOID * pvOut,
    LPUNKNOWN pUnkOuter
);
```

メソッドでデバイスを作成する。rguidにはキーボードを使うならGUID_SysKeyboardを，マウスならGUID_SysMouseを，ジョイスティックなら先ほど取得したguidJoystickを使用する。riidはIID_IDirectInputDevice2を，pvOutには取得したいデバイスのポインタを渡す。最後はNULLでいい。そうしたら，入力するデータのフォーマットを設定する。

```
IDirectInputDevice7::SetDataFormat(
    LPCDIDATAFORMAT lpdf
);
```

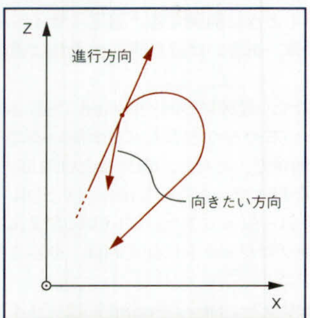


図12　1パスだと大きく外れることがある

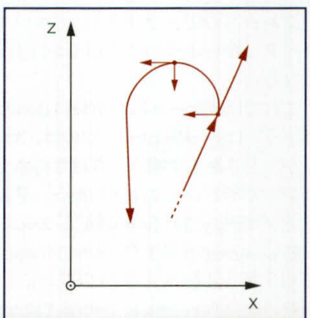


図13　画面中心を経由して２パスで振り向かせる

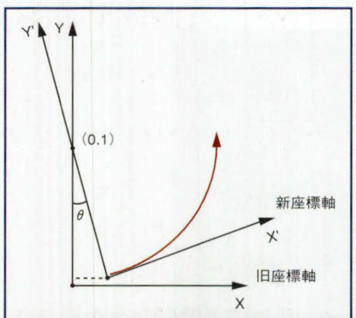


図14　螺旋運動する軸のTranslation

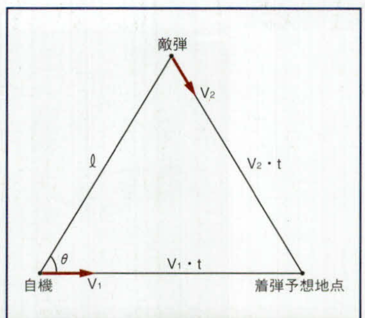


図15　着弾予想

引数は，デバイスが一般的なキーボード，マウス，ジョイスティックでいい場合は，それぞれc_dfDIKeyboard，c_dfDIMouse，c_dfDIJoystickのアドレスでいいようだ。これらはライブラリ内で宣言，確保されているらしい。

```
IDirectInputDevice7::SetCooperativeLevel(

    HWND hwnd,
    DWORD dwFlags
);
```

でDirectDraw同様に協調レベルを設定，最後にIDirectInputDevice7::Acquire()でデバイスのアクセス権を取得して，初期化は完了である。ジョイスティックの場合は，範囲などのプロパティの設定も必要だが，それらについてはソースを見てほしい。

実際のデータの取得には，

```
IDirectInputDevice7::GetDeviceState(
    DWORD cbData,
    LPVOID lpvData
);
```

を使う。2つ目の引数は，デバイスによって渡すものが違い，キーボードの場合は256個のBYTE配列を，マウスの場合はDIMOUSESTATE構造体を，ジョイスティックの場合はDIJOYSTATE構造体となる。ひとつ目の引数はそのサイズだ。もしキーなりボタンなりが押されていれば，配列あるいは構造体にそれに対応するデータが格納される。たとえばジョイスティックのひとつ目のボタンが押されていれば，DIJOYSTATE構造体のメンバrgbButtons[0]の上位ビット（8ビット）がセットされる。

プログラムでは，F1キーでキーボード，F2キーでマウス，F3キーでジョイスティックに入力を切り替えられるようにした（ファンクションキーの入力自体は，WM_KEYDOWNメッセージで受けている）。まあでも結局のところ，キーボードで遊ぶのがいちばん簡単なような気がする。

ここまでがサンプルgame2である（**図16**）。game1でつけていたミサイル積載カメラは外し，量産型Demetriusの硬さは3に設定されている。また，追尾ミサイルの最大回転角は大きめに，一度のロックで敵1機あたりに発射する弾数は3にしてある。この辺は，撃破した敵の数に応じてパワーアップさせる予定だ。さらに，追尾ミサイルは敵が画角を外れても，半分の確率で見失わないようにしている。こうすることで，敵にかわされても周囲をくるくるっと回ってまた敵に当たったりするようになる。まあ，要するにビジュアル的な要素だ。自機のシールドやスコアも表示するようにしたが，まだ自分は死ぬことはない。フレームレート調整ウェイトも入れたので，速すぎて遊べないということもないだろう。



図16　サンプルgame2

●敵マップを作る

ここまでは敵はランダムに出現していた。それはそれでそーゆーゲームもあるんだが，このゲームでは敵のマップを作って，敵の出現に再現性を持たせることにする。マップは，グラフィックツールなどで敵の種類ごとに色を決め，BMPにしたあとになんらかのかたちにコンバートしようかと思ったのだが，コンバータを作っている時間が惜しかったので，手で書いてしまった。フォーマットは，フレームナンバー，X座標，Y座標，敵のタイプの順で格納したshort配列である。Z座標は画面いちばん奥150（ここまでは100であったが，ちょっと浅かったので150にした）である。ここに，ボスの出現ポイントまで格納してある。ザコの数は一面あたり100，3ステージ分用意した。つまり，全3面ってことだ。ただし，ボスとの戦闘中は，ザコを適当な確率で出現させるようにしている。

ここでやっとボスの登場だ。1・2面目はDemetrius，3面はEresh-ki-galである。硬さはそれぞれ40と80。ただし，1面目のDemetriusは半分，つまり20発の弾を食らうと逃げてしまうようにしてある。モーションについては量産型Demetriusに似ている。一定距離まで近づくと停滞して弾を撒き，反転・離脱後に再び旋回してやってくる。

ただし，ここで後ろに抜けていくのではなく，また停滞して弾を撒くようにしている。その繰り返しだ。また，近づいてくるときには，直線ではなく常にこちらに向きを合わせようと角度を調整，さらに軸線上に乗ったら，ゆっくりと螺旋運動するようにしてある。これは，真正面からまっすぐ飛んできたときに，こちらの通常弾をまともに食わないようにするためである（通常弾は回避運動はしない）。

さて，このモーションをつけたとき，撃墜していないのに，ごくまれに突然ボスが消えてしまうというバグが発生した。当初は原因がまったくわからずに，しかも発生頻度が極端に低いこともあって，これによって2000年の三が日を頭を抱えて過ごす結果となった。なかなかに不幸な出来事である。その原因となっていたのは，角度の算出部分であった。

```
D3DRMVectorCrossProduct( &vr, &v1, &v2 );
D3DVALUE len = D3DRMVectorModulus(&vr);
D3DVALUE th = asin( len );
```

v1は的の進行方向，v2は自機への向きである。それの外積を取ることで，敵の旋回方向を得ようとしていたのだが，2つのベクトルが90度をなすとき，外積によって得られたベクトルの長さ，つまり上でいうところのlenが，誤差により1.0よりわずかに大きくなることがあるようなのだ。asin()が引数として取れるのは－1.0から1.0の範囲で，それ以外だと不定数が返る，つまりthは不定となり，それ以降の結果もまったく予測できない結果となる。これはボスに限った話ではないのだが，ボス以外ではそのようなことがあっても気づかなかったのだろう。このような事態を避けるために，lenが1.0よりも大きい場合のチェックはしっかりとやらなければならない。

最後はファンネル。こいつらはEresh-ki-gal側で管理するのではなく，1つひとつを独立した敵にした。そうしておいて，おのおのでEresh-ki-galのフレームの周りを浮遊し，ついていくように制御する。追尾ミサイルはロックしないようにしてあるが，通常弾，あるいは流れ弾が当たれば破壊される。

これでほぼゲームはかたちになったので，最後にDirectSoundでSEをつけよう。DirectSoundってのは，いろいろやろうと思えば奥が深いのだろうが，とりあえず鳴らすだけなら割と簡単だ。ただし，爆発音だけは3Dサウンドでポジショニングしたい。そんなわけで，こちらもDirectX SDKのサンプルPlay 3D Soundからソースを引っ張ってきた。こいつに含まれているwavread.hおよびwavread.cppをプロジェクトに加えれば，やることはそう多くはない。

その前にDirectSoundの構造について少々。DirectSoundにはプライマリバッファとセカンダリバッファがある。プライマリバッファとは，サウンドカードが直接データを読み出すバッファだと思えばよい。ユーザーは普通

これを直接いじることはない。それに対し，セカンダリバッファとは，再生したいサウンドデータを保持しておくバッファである。ユーザーはこちらのバッファに対して制御を行う。たとえばセカンダリバッファに対して再生命令を出せば，プライマリバッファにデータがコピーされて再生される。また，セカンダリバッファは複数確保することができ，同時に再生しても自動的にミキシングされてプライマリバッファに格納される。ただし，あくまでもひとつのセカンダリバッファで1声だ。同じ音を複数重ねたい場合も，それだけセカンダリバッファが必要となる。また，DirectSoundは3Dサウンドもサポートする。これは音源の位置（とリスナーの位置）を指定することで，パンや減衰，ドップラー効果などをシミュレートしてくれる。

初期化の仕方なんかはほかと同じようなもんなんで，くどくど説明せずに省略。サウンドの読み込みからにしよう。用意したサウンドファイルは次のとおり。

リニアカノン発射音
追尾ミサイルロックオン
追尾ミサイル発射音
自機のダメージ音
爆発音

これらのWaveファイルを西川善司氏に作ってもらった。ただし，DirectSoundはWaveファイルをサポートしない。そこで必要になってくるのが，先ほどプロジェクトに加えたwavread.hとwavread.cppで定義されているCWaveSoundReadクラスだ。このクラスのインスタンスを作成してOpen()メソッドでファイルを読み込み，Read()メソッドでセカンダリバッファに流し込む。この辺は同じくサンプルから引っ張ってきたCreateStaticBuffer()とかFillBuffer()関数をほとんどそのまま使える。ただし，爆発音以外は3Dである必要はない。また，爆発音は3声くらいはほしい。そのためには，爆発音のセカンダリバッファは3つ必要だが，IDirectSound::DuplicateSoundBuffer()を使えば，同じバッファを共有するセカンダリバッファを作ることができる。しかし，バッファがハードウェア上に取られている場合など，失敗することもあるらしいので，その場合はほかと同じように確保させることも忘れてはならない。

3Dサウンドを使うには，IDirectSound3DBufferというインタフェイスを使う。これは，セカンダリバッファであるIDirectSoundBufferからQueryInterface()で取得できる。こいつのインスタンスに対して，

```
IDirectSound3DBuffer::SetPosition(
    D3DVALUE x,
    D3DVALUE y,
    D3DVALUE z,
    DWORD dwApply
);
```

メソッドを呼ぶことで，音源のポジションを設定できる。おわかりのように，座標はD3DVALUEで記述されている。つまりDirect3Dとの併用が考慮されており，いまの例でいえば，爆発の起こった座標をそのまま入れればいいことになる。ちなみに最後の引数は，すぐにこの設定を反映させるかどうかを示している。

DirectSoundはこういったパラメータの変更があった場合，プライマリバッファの再レンダリングが行われる。しかし，これと同時にほかの変更，たとえばリスナーの座標の変更を行った場合など，再びレンダリングが行われ，二度手間になる。そこで，ここでDS3D_DEFERREDを指定しておけば，IDirectSound3DListener::CommitDeferredSettings()が呼ばれるまでは設定を反映しない。今回はプレイヤーは常に原点に設定されているので，リスナーは作らず，ここでの引数はすぐに設定を反映するDS3D_IMMEDIATEを指定した。

そんなわけで最終的に完成したのが，サンプルgame3，正式名SION2000である（図17）。結局そういうことになってしまった。こんな中



図17　サンプルgame3 SION2000

途半端なゲームにSIONの名をつけちゃって心苦しくはあるのだが，山田氏も許してくれるよね。敵を120機落とすごとにレベルアップし，最初は追尾ミサイルが敵1機に対して1発だったのが，レベル2で2発，レベル3からはミサイルの最大回転角が大きくなり，レベル4で3発となる。一応おまけ程度のスタート画面やエンディングもつけてみた。そうそう，ステージごとに背景を変え，ステージ2ではガス面にした。この面はフォグがかかっており，遠くの敵は見えにくくしてある。あと，細かい点を修正してゲームバランスを調整したつもりなんだが，結構簡単になってしまったかもしれない。ボスだけがやたら硬いってのは避けたかったんだけど，ザコが考えなしすぎたのかも。

●ちょっと重かったね

描画じゃなくて，演算が少々ヘビーだったようだ。特に追尾ミサイル。毎フレームあれだけ真面目に計算してりゃ，そりゃそうか。たとえば4フレームごとくらいに計算して，あとは惰性で動かすとかすればよかったのかもしれない。そうすれば，全体的にもうちょっとタイトにもなったし。で，トリプルバッファにでもして，フレームレートのばらつきを吸収するとか。とりあえず今回は実験的な部分もあったんで，かたちになった点でよしとしようか。

Column

●ところでSIONってなに？

旧Oh!X読者ではなかった方のために解説しておこう。SIONは3Dグラフィックパッケージ MAGIC（吉村氏の開発によるZ80用の3Dライブラリ）をX68000に移植したもの（最初はハンドコードコンバートと描画はIOCSコールそのままというちょっと乱暴なものだった）のサンプルとして「やっぱ，ゲームでしょ」ということで山田純二氏が作成した3Dシューティングゲームだ。

その後，MAGICのバージョンアップが進み，編集部の総力を挙げてSION IIが開発される。ちゃんとバランス調整をしたのでそこそこいいゲームに仕上がったとは思う。

艦隊戦シミュレーションタイプのSION IIIは開発が中断のため欠番。ビジュアル面では視点ぐりぐりのアコンカグヤレーザービシバシのかなりカッコいい仕上がりだったのだけど，ゲーム本体が進まなかったようだ。

そして丹/横内コンビによるポリゴンパッケージSLASHのサンプルとしてSION IVが制作される。システムやエフェクトなど，技術的には頂点といっていい作品だが，開発期間バランス調整はいまひとつのままだったのが惜しまれる。まあ，編集者がゲーム作るってのに無理はあったのだが。

今回のSION2000はSION IVっぽい作品ということで制作されたものだが，完成度はかなり劣るので注意。

DirectXで手軽にゲームを作ってみよう
現代版テーブルテニス "THE GARAGE STONES"

飯田　伸一　Iida Shinichi

最近のゲームというと，3Dバリバリの豪華なものという印象が強くなりすぎてる感がある。しかし，シンプルなものにも味わいはある。面白さの本質的な部分はむしろ単純なゲームでこそ引き立ってくる。ローファイ精神で彩られた古典的ゲームであるテーブルテニスをお届けしよう。

特集がゲーム制作系ということで，テーブルテニス（のようなもの）を作ってみた。ご存じの方も多いとは思うが，このテーブルテニスは，ゲーム界の三葉虫と形容されるほど，とても古い歴史を持つゲームだ。かつてのマイコン全盛時代に（思わず遠い目……）構造が比較的シンプルということで，主にBASICなどの言語の入門用に，いわゆる「定番」モノとしてよく作られたりしたものだ。これを今回のゲーム制作系の題材として，ここで取り上げてみることにしよう。さっそくゲームの解説に取りかかりたいところだが，その前にどうして僕が上記の理由に加え，このようなシンプルなゲームを取り上げるのかということについて，多少ゴタクを述べたい。

ローファイ精神

現在の音楽の世界，とりわけロック（こういう分類も，次第に意味を失ってきているのだが……）では，高価な機材や高度な技術を用いなくても，音楽的に面白いものを作るアーティストが増えている。

彼らの大半は，演奏はヘタクソだし，おまけにボーカルが音痴で，音を豪快にはずしても平然となに食わぬ顔（！）をしていたりする豪傑も少なくない。

それに加え，3コード程度でも組み合わせによっては，どんなことでも表現できると彼らから学んだときは，とかく無意味に高度な技巧に走りたがるマッチョな体育会系メタルなんて下らなく思えてしまったほどだ。

音楽的なことは別としても，思想的な影響はかなり受けた。彼らの「自分でやろう」，「自分自身でいよう」というローファイ（ローテク）精神は，作り手と受け手を阻んでいる距離を縮め，お互いのミゾを埋めて，楽しもうというグルーヴを生み出した。

これに触発された僕は，高度な技術を用いないで，ひとりでどこまで面白いゲームが作れるかをとことんまで追求したくなった（現在，多数出まわっている見栄えだけよくて，つまらないゲームへのアンチテーゼという意味も含めて）。一見地味なローファイ（Low Fidelity）なゲームでも，作り方によっては，とても面白いものができることを知ったからだ。[*1]

ローファイなミュージシャンたちのなかには，過去の作品をコラージュ，サンプリングといった手法を用いて，新たな作品（？）を作り出している者も多い。今回僕は，テーブルテニスを自分なりに再解釈し，アレンジを施してみたというわけだ。

高度化したゲーム制作環境への異議申し立て

僕の経験からいうと，初心者がゲームを作ろうとしても，ゲームの高度化にともなって開発環境は複雑化しており，覚えなくてはならない（あるいは覚えなくてもいい）ことがあまりにも多すぎるのだ。その結果として，ゲーム制作の入り口までたどりつかずに諦めてしまう人が非常に多いという，とても不幸な状況がそこにはある。パソコンが世の中にどんどん浸透するにつれて，以前からは信じられないほど，ハードの価格は下がり，性能はとてつもなく向上したにもかかわらずだ。

最初から，ポリゴンだかポリヘドロン（これはないか……）を駆使したゲームなんて作れるはずはない。まずは，このようなシンプルなゲームから作り始めるべきなのだ。構造こそ単純だが，ゲーム作りに必要なエッセンスがつまっている。逆にいうと，自分でもこのようなシンプルなゲームが作れるようになれば，あとは自然と自分になにが足りないかを知ることができるので，その不足している部分を少しずつ補ってやればいいだけなのだ。ここまでくれば，あとは知らないうちに，ゲーム制作のノウハウが蓄積され，作りたいものは自分で作れるレベルにあっというまに達するはずだ（よね……）。

要するに，なぜ僕がこのゲームを取り上げるかというと，つまるところ高度化しているゲーム制作環境への異議申し立てにほかならない。

手軽にゲームを作るために……

現在，パソコンでゲームを制作する場合はDirectXを使用するのが一般的だ。しかし，DirectXは，その機能が複雑であり，手軽になにかを作ろうとしても，なかなか煩雑で面倒なものになっている（もともと開発者向けに作られたものだからしかたがないのだが）。とりわけ，ゲーム開発の初心者には乗り越えなければならない大きな壁となっている。そこで，ここでもDirectXの煩雑な機能を簡単に利用できるBotchy氏作のフリーライブラリ"el"を使わせていただいた（elについての詳細は，ヘルプファイルを参照してほしい）。

このelを使用することによって，簡単なC言語の知識だけで，ゲームが作れるようになる。ちなみにC言語について，理解が不十分だと感じている方

＊1　少し表現が過激なのは，ガッツ石松氏著作の玉稿に感動したから……ではない。

は，連載の「ピコピコC言語」の本文と，そのおまけのリファレンス（もどき）をあわせて読んでいただきたい。これでelの仕組みとC言語について，大まかな理解は得られるはずだ（これで三方丸く収まるという大岡裁き……）。
※elはCD-ROMに収録。

ゲームの解説

テーブルテニス〝THE GARAGE STONES〞（画面1，リスト1）のルールは，お互いにボールを打ち合って，相手側のコートの中にボールを入れれば，点数が加算され，先に5点を先取したほうが勝ち，という簡単なものだ。ただ，前述の理由のとおりに，ただ単純に昔そのままのテーブルテニスを再現したのではなく，ゲームセンターによくあるようなホッケーゲームのように，プレイヤーが操作するパドルは，左右だけではなく，上下にも動けるようにした。つまり，ボールが移動中にパッドで，上（または下）方向に打ち返してやると，ボールの移動速度を速くしたり，逆に遅くしたりすることができるのだ。

プレイヤー1は，〝WASD〞キーで，プレイヤー2はカーソルキーでパドルを操作する（それぞれジョイスティックにも対応している）。ゲーム終了はエスケープキーだ。なお，凶悪なコンピュータ対戦モードもあるので，ほかのプレイヤーと対戦する前に練習しておこう。コンピュータ対戦モードの強さは，ファンクションキーのF1-F3（イージー，ノーマル，ハード）で変更することができる。

改造，解析のススメ

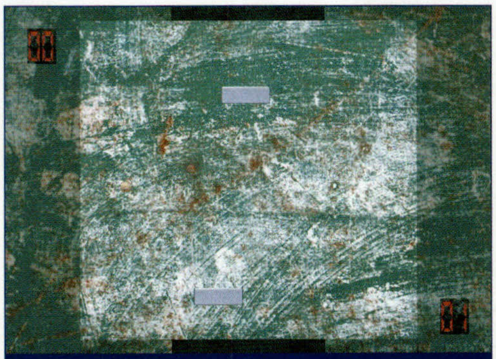
このゲームはただ受動的に遊ぶだけではなく，グラフィックを入れ替えてみたり，プログラムに自分なりに手を加えてみてほしい。僕が古典的なテーブルテニスに手を入れて，新たなゲームを制作したのと同様に，このゲームを叩き台にして，いろいろな機能を追加したり，あるいはまったく別なゲームを作る際の素材（のようなもの）として使ってもらえれば幸いだ。あ，書き忘れたけれど，このゲームはコピーレフト（著作権フリー）だ。

グラフィック，サウンド関連

Windows付属の〝ペイント〞や，各自所有しているグラフィックツールで，このゲームで使用しているグラフィック関連が格納されているフォルダ"Bmp"の中のバンプマップファイル（xxx.bmpとなっているもの）を呼び出してみよう。このグラフィック周りを少しでも自分なりに書き換えてみることによって，このゲームの持つ雰囲気が意外と大きく変わるのがわかってもらえるだろう。

サイズはそれぞれ，ボールは16×16，パドルは64×24，背景は640×480ドットだ。

ちなみに，今回このゲームを制作するのに当たって，土井淳氏作のフリーのグラフィックツール"D-Pixed"を使わせていただいた（画面2）。このソフトはフリーながら，高機能でとても使いやすいものになっている（特にスプライト作成時に威力を発揮する）。ゲーム制作に最適なグラフィックツールのひとつであると思う。今回のCD-ROMではTOOLSディレクトリに収録されているので，ぜひ試してもらいたい（なによりもタダというのが嬉しい……）。



画面1　対戦型テーブルテニスだ

リスト1

```
/***************************************************************************/
/*                                                                         */
/*  "THE GARAGE STONES"                                                    */
/*                                                                         */
/***************************************************************************/

// 以下のコメントをはずすと，" Borland C++ Builder3" でコンパイルが可能になる
//#define BCB WinMain

#include "el.h"

#define GAMESET 5  // ゲームセット数

// コンピュータの難易度
#define EASY F(280)
#define NOMAL F(320)
#define HARD F(340)

// 画面定義
#define LOAD_SCREEN 1  // ロード画面
#define TITLE_SCREEN 2  // タイトル画面
#define MAIN_SCREEN 3  // メイン画面

// 画面用関数のプロトタイプ宣言
void LoadScreen(void);
void TitleScreen(void);
void MainScreen(void);

// 各関数のプロトタイプ宣言
void InitFunc(void);  // 初期化
void OverFunc(void);  // ゲームオーバ
void BlinkFunc(int);  // スコア点滅

// グラフィックオブジェクト
DDOBJ PadBMP;      // パドル
DDOBJ BallBMP;     // ボール
DDOBJ ScoreBMP;    // スコア
DDOBJ CanvasBMP;   // 背景
DDOBJ ScreenBMP;   // 背景保存用

// サウンドオブジェクト
DSOBJ HitWAV; // 命中
DSOBJ MissWAV;// ミス
DSOBJ BallWAV;// ボール

// ボール
struct ball{
   bool Vw;          // 描画フラグ
   float X,Y;        // 座標
   float XAdd,YAdd; // 移動量
   float AddSpeed;  // 速度
}Ball;

// パドル
struct pad{
   bool Vw;          // 描画フラグ
   bool Rd;          // 待機フラグ
   int Score;        // スコア
   int JX,JY;        // ジョイスティック
   int Up,Down,
       Left,Right;  // キー入力
   float X,Y;        // 描画座標
   float XAdd,YAdd; // 移動量
   float Hit;        // 命中
}Pad[2];

// ゲーム情報
bool InitData;// 初期化フラグ
bool GameOver;// ゲームオーバフラグ
bool BlinkFlag;   // 点滅フラグ
```

```
bool Com;      // コンピュータ対戦フラグ
float ComLv;   // コンピュータの強さ
int Service;   // サービス

// キー操作情報
const int Key[2][4]={
   'W','S','A','D',                  // Player1
   VK_UP,VK_DOWN,VK_LEFT,VK_RIGHT // Player2
};

// 反発力テーブル
float PadDir[16][2]={
       -257.082106f,-138.905142f,     // 左
       -257.082106f,-147.085399f,
       -264.576387f,-141.419023f,
       -231.903130f,-190.317982f,
       -190.317982f,-231.903130f,
       -141.419023f,-264.576387f,
       -87.0853995f,-287.082106f,
       -29.405142f,-298.555416f,      //
        29.405142f,-298.555416f,      // 中央
        87.0853995f,-287.082106f,
        141.4190235f,-264.576387f,
        190.3179825f,-231.903130f,
        231.9031305f,-190.317982f,
        264.5763873f,-141.419023f,
        257.0821063f,-147.085399f,
        257.0821064f,-138.905142f     // 右
};

// 汎用
int i,j,x,a;
char Buff[32];

// メイン関数

int elMain("THE GARAGE STONES")
{

// ウィンドウモードで動作させたい場合は,
// 以下のコメントをはずす
// elWindow(640,480,FALSE);
// elSetWindow(RUN_NOTOP|RUN_INACTIVE);

   elLoop()
   {
       elSetScreen(LOAD_SCREEN,LoadScreen());
       elSetScreen(TITLE_SCREEN,TitleScreen());
       elSetScreen(MAIN_SCREEN,MainScreen());
   }

   elExitMain();
}

// ウィンドウ生成関数

void elCreate(void)
{
   elDraw::Screen(640,480);
   elCallScreen(LOAD_SCREEN);
}

// キーボード関数

void elKeyboard(void)
{

   // スペース, リターンキー押下による1P,2Pモードの切り替え
   case VK_SPACE:
   {
       Com=TRUE;
       break;
   }

   case VK_RETURN:
   {
       Com=FALSE;
       break;
   }

   // ファンクションキー押下による難易度の切り替え
   case VK_F1:
   {
       ComLv=EASY;
       break;
   }

   case VK_F2:
   {
       ComLv=NOMAL;
       break;
   }
   case VK_F3:
   {
       ComLv=HARD;
       break;
   }

   // エスケープキーが押された場合
   case VK_ESCAPE:
   {
       if (elGetScreen()==TITLE_SCREEN)
       {
           // ゲーム終了
           elDraw::Exit();
       }
       else
       {
           // タイトル画面へ戻る
           elCallScreen(TITLE_SCREEN);
       }

       break;
   }

   elExitKeyboard();
}

// イベント関数

long elEvent(void)
{
   elExitEvent();
}

// ロード画面

void LoadScreen(void)
{
   // ビットマップファイルの読み込み
   BallBMP=elDraw::LoadObject("BMP\\BALL.BMP");
   PadBMP=elDraw::LoadObject("BMP\\PAD.BMP");
   CanvasBMP=elDraw::LoadObject("BMP\\CANVAS.BMP");
   ScoreBMP=elDraw::LoadObject("BMP\\SCORE.BMP");

   // サウンドファイル(WAVE)の読みこみ
   HitWAV=elSound::LoadObject("SOUND\\HIT.WAV");
   MissWAV=elSound::LoadObject("SOUND\\MISS.WAV");
   BallWAV=elSound::LoadObject("SOUND\\BALL.WAV");

   // 背景をゲーム用に加工
   elDraw::Layer(0,0,CanvasBMP,0,0,640,480);

   elDraw::DelBlendLayer(220,456,CanvasBMP,220,456,420,480);
   elDraw::DelBlendLayer(220,0,CanvasBMP,220,0,420,24);
   elDraw::DelBlendLayer(32,34,CanvasBMP,18,0,56,48);
   elDraw::DelBlendLayer(570,402,CanvasBMP,18,0,56,48);

   // 加工した背景を新たなスプライトとして保存
   ScreenBMP=elDraw::CreateObject();
   elDraw::SpriteScreen(ScreenBMP,0,0,640,480);

   elCallScreen(TITLE_SCREEN);
}

// タイトル画面

void TitleScreen(void)
{
   elDraw::Clear();

   // タイトル画面の描画
   elDraw::Layer(0,0,CanvasBMP,0,0,640,480);

   elFont::Begin(GOTHIC,70,0,TRUE);
   elFont::Color(RGB(148,156,189));
   elFont::Draw(200,90,"THE");
   elFont::Draw(200,150,"GARAGE");
   elFont::Draw(200,210,"STONES");
   elFont::Before();

   elFont::Begin(GOTHIC,10,0,TRUE,FALSE,FALSE);
   elFont::Color(RGB(220,220,220),RGB(10,10,10),FALSE);
   elFont::Draw(101,24," Lo-Fi_Teens ");

   elFont::Color(RGB(50,50,50),RGB(220,220,220),FALSE);
   elFont::Draw(389,446," Space(1P)  Enter(2P)  Esc(X) ");
   elFont::Before();

   elDraw::Refresh();

   // キー入力があった場合
   if (PushKey==VK_SPACE || PushKey==VK_RETURN){
       Service=2;
```

```
        Pad[0].Score=0;
        Pad[1].Score=0;
        ComLv=NOMAL;
        elCallScreen(MAIN_SCREEN);
    }
}

// メイン画面

void MainScreen(void)
{
    // 画面が切り替わった直後の場合
    if (elChangeScreen())
    {
        // 初期化処理関数の呼び出し
        InitFunc();
    }
    elDraw::Clear();

    // ゲーム画面の描画
    elDraw::DirectLayer(0,0,ScreenBMP,0,0,640,480);

    for (i=0;i<2;i++)
    {
        // キー入力部
        elSystem::GetKey(Key[i][0],&Pad[i].Up);
        elSystem::GetKey(Key[i][1],&Pad[i].Down);
        elSystem::GetKey(Key[i][2],&Pad[i].Left);
        elSystem::GetKey(Key[i][3],&Pad[i].Right);
        elJoystick::Stick((i? (JOY2) : (JOY1)),&Pad[i].JX,&Pad[i].JY);

        // 方向キーが押されている場合
        if (Pad[i].Left==HOLD_KEY || Pad[i].JX==-1)
            Pad[i].XAdd=F(-400);

        if (Pad[i].Right==HOLD_KEY || Pad[i].JX==1)
            Pad[i].XAdd=F(400);

        if (Pad[i].Up==HOLD_KEY || Pad[i].JY==-1)
            Pad[i].YAdd=F(-200);

        if (Pad[i].Down==HOLD_KEY || Pad[i].JY==1)
            Pad[i].YAdd=F(200);

        // 何もキー入力がない場合
        if (Pad[i].Left==FREE_KEY &&
            Pad[i].Right==FREE_KEY && Pad[i].JX==0)
            Pad[i].XAdd=F(0);

        if (Pad[i].Up==FREE_KEY    &&
            Pad[i].Down==FREE_KEY && Pad[i].JY==0)
            Pad[i].YAdd=F(0);

        // 対コンピュータの移動処理
        if (Com && Ball.Vw){
            if (Ball.X>Pad[1].X-F(32))
                Pad[1].XAdd=ComLv;

            if (Ball.X<Pad[1].X+F(32))
                Pad[1].XAdd=-ComLv;

            if (Ball.Y<F(300)){

                if (Ball.YAdd<F(0))
                    Pad[1].YAdd=F(150);

                if (Ball.YAdd>F(0) || Ball.Y<Pad[1].Y)
                    Pad[1].YAdd=F(-150);
            }
        }

        // パドルの移動処理
        Pad[i].X+=FrameTime*Pad[i].XAdd;
        Pad[i].Y+=FrameTime*Pad[i].YAdd;

        // パドルの移動範囲をチェック
        if (Pad[i].X<F(100))
            Pad[i].X=F(100);

        if (Pad[i].X>F(640-64-100))
            Pad[i].X=F(640-64-100);

        if (Pad[0].Y<F(240+48))
            Pad[0].Y=F(240+48);

        if (Pad[0].Y>F(480-24-70))
            Pad[0].Y=F(480-24-70);

        if (Pad[1].Y<F(70))
            Pad[1].Y=F(70);

        if (Pad[1].Y>F(240-24-70))
            Pad[1].Y=F(240-24-70);

        // パドルの描画
        if (Pad[i].Vw)
            elDraw::Layer((int)Pad[i].X,(int)Pad[i].Y,PadBMP,0,0,64,24);
        // ボールに当たっている場合のパドルの描画
        else{
            elDraw::Layer((int)Pad[i].X,(int)Pad[i].Y,PadBMP,65,0,128,24);

            Pad[i].Hit+=FrameTime*F(100);

            if (Pad[i].Hit>=F(15)){
                Pad[i].Vw=TRUE;
                Pad[i].Hit=F(0);

                elSound::Play(HitWAV);
            }
        }

        // パドルとボールの当り判定
        if ((int)Ball.Y>Pad[i].Y-16 && (int)Ball.Y<Pad[i].Y+24){
            if ((int)Ball.X>Pad[i].X-8 && (int)Ball.X<Pad[i].X+64-8){

                // パドルのどこにボールが当たったのか？
                x=((int)(Ball.X+8-Pad[i].X)>>2)&0x0f;

                // 反発力テーブルを参照して，ボールの移動量を決定
                Ball.XAdd=PadDir[x][0];
                Ball.YAdd=PadDir[x][1];

                // パドル操作によるボールの移動速度の変化
                if (Pad[i].YAdd<F(0))
                    Ball.AddSpeed=F(1.5);
                    else if (Pad[i].YAdd>F(0))
                        Ball.AddSpeed=F(1);

                if ((int)Ball.Y>Pad[i].Y){
                    Ball.YAdd*=-1;
                    if (Pad[i].YAdd<F(0))
                        Ball.AddSpeed=F(1);
                        else if (Pad[i].YAdd>F(0))
                            Ball.AddSpeed=F(1.5);
                }

                Ball.XAdd*=Ball.AddSpeed;
                Ball.YAdd*=Ball.AddSpeed;

                Pad[i].Vw=FALSE;
            }
        }

        // スコア表示
        if (!Pad[i].Rd){
            sprintf(Buff,"%02d",Pad[i].Score);

            for(j=0;j<2;j++){
                if (Buff[j]==' ') continue;
                    a=Buff[j]-'0';

elDraw::Layer(32+(i*538)+(j*18),32+(i*368),ScoreBMP,a*18,0,a*18+18,48);
            }
        }
    }

    // ボールの移動処理＆描画
    if (Ball.Vw){
        Ball.X+=FrameTime*Ball.XAdd;
        Ball.Y+=FrameTime*Ball.YAdd;
        elDraw::Layer((int)Ball.X,(int)Ball.Y,BallBMP,0,0,16,16);
    }

    // ボールの移動範囲をチェック
    if (Ball.X<F(100)){
        Ball.X=F(100);
        Ball.XAdd=-Ball.XAdd;
    }

    if (Ball.X>F(524)){
        Ball.X=F(524);
        Ball.XAdd=-Ball.XAdd;
    }

    if (Ball.Y<F(24)){
        if (Ball.X<F(220-8) || Ball.X>F(420)){
            Ball.Y=F(24);
            Ball.YAdd=-Ball.YAdd;
        }

        // Player2がミスをしたか？
        if (Ball.Y<F(14)){
            Service=1;
            InitData=TRUE;
```

```
        }
    }

    if (Ball.Y>F(480-32)){
        if (Ball.X<F(220-8) || Ball.X>F(420)){
            Ball.Y=F(480-32);
            Ball.YAdd=-Ball.YAdd;
        }

        // Player1がミスをしたか？
        if (Ball.Y>F(480-24)){
            Service=0;
            InitData=TRUE;
        }
    }

    if (InitData){
        Pad[Service].Score++;
        elSound::Play(MissWAV);
        InitFunc();
    }

    if (GameOver)
        OverFunc();

    // スコア点滅
    if (BlinkFlag)
        BlinkFunc(Service);

    elDraw::Refresh();
}

// ゲーム初期化関数
void InitFunc(void)
{
    // プレイヤー情報の初期化
    InitData=FALSE;
    GameOver=FALSE;

    for (i=0;i<2;i++){
        Pad[i].Vw=TRUE;
        Pad[i].Hit=F(0);
        Pad[i].X=F(290);
        Pad[i].Y=F(416)+i*(-346);

        // ゲームセットの場合は，フラグを立てる
        if (Pad[i].Score>=GAMESET)
            GameOver=TRUE;
    }

    // ボール情報の初期化
    Ball.Vw=FALSE;
    Ball.X=F(210);
    Ball.Y=F(240);
    Ball.XAdd=F(100);
    Ball.YAdd=F(100);
    Ball.AddSpeed=F(1);

    if (Service==1)
        Ball.YAdd=-Ball.YAdd;

    // 点滅フラグを立てる
    BlinkFlag=TRUE;

    elSystem::SetWait(1,500);
}

// ゲームオーバ関数
void OverFunc(void)
{
    Service=2;
    elFont::Begin(GOTHIC,24,0,TRUE);
    elFont::Color(RGB(220,220,220),RGB(10,10,10),FALSE);
    elFont::Draw(296,240,"OVER");
    elFont::Before();

    // スコアの点滅が終わったら，タイトル画面へ
    if (!BlinkFlag)
        elCallScreen(TITLE_SCREEN)
}

// スコア点滅関数
void BlinkFunc(int m)
{
    static int c=7; // 点滅カウンタ

    if (elSystem::Wait(1)){
        elSystem::InitWait(1);
        elSystem::SetWait(1,500);

        if (m==0 || m==2)
            Pad[0].Rd=!Pad[0].Rd;
        if (m==1 || m==2)
            Pad[1].Rd=!Pad[1].Rd;

        if (!c){
            Pad[0].Rd=FALSE;
            Pad[1].Rd=FALSE;
            BlinkFlag=FALSE;
            c=7;

            if (!GameOver){
                elSound::Play(BallWAV);
                Ball.Vw=TRUE;
            }
        }
        c--;
    }
}
```

※D-Pixedの作者，土井淳氏のホームページ，
"http://www.win.ne.jp/~doichan/"

効果音（WAVE）は，いまひとつ自分でも納得していないので，各自サウンドツールで，いろいろ加工してみたり，あるいはまったく新しい音を作成したりと，差し替えてみていただきたい。あとMIDIでお好みのBGMを流すのも，いいかもしれない。

プログラムについて

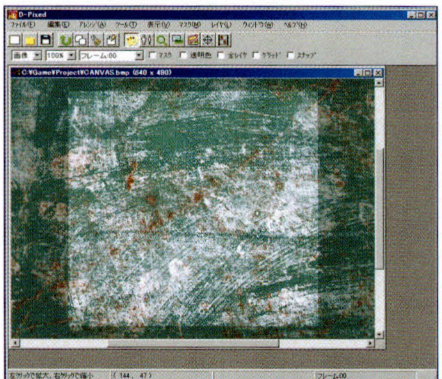
例によってプログラム中では，難しいことはそんなにしていないが，他人が作ったプログラムを解析することは，その人なりクセとかがあり，スムーズに読み進めない場合が往々にしてあるものだ。以下に，簡単ではあるが，プログラムを読むに当たってのポイントのようなものを記したい。

elというライブラリは，ゲーム画面ごとに関数を用意する（タイトル画面，メイン画面など）。そして，指定した画面関数を何回もループさせてやることによって，プログラムが進行していく。必然的に，メイン画面関数に処理が集中することになるので，なにかイベントが起きたときには，フラグを用いてそのイベントの処理をする。

たとえばこのゲームにおいては，パッドにボールが当たったときは，そのフラグの判定に従って，パッドを光らせたり，効果音を出したりしている。ここのところが，いままでのプログラミングスタイルとは少し違うので注意してほしい。

・ロード画面関数，"LoadScreen()"

この関数では，ゲームで使用するビットマップファイル，サウンドファイル（WAVE）の読み込みをしている。ここでほかに行っている重要な処理



画面2　D-Pixedでの背景編集

は，ゲーム用に背景を加工しているところだ。具体的には，各プレイヤーの得点表示部分とボールが入ったらミスになる部分を50％減算アルファブレンド（暗めに描画すること）して，新たなグラフィックオブジェクト"Screen BMP"に保存している。

なぜ，ゲーム用にもうひとつ別にグラフィックオブジェクトを用意しているのかというと，アルファブレンドはやや重い特殊効果なので，ゲーム中に毎回この処理で書き換えを行うと，ゲーム全体の処理速度が落ちてしまうからなのだ。

・タイトル画面関数，"TitleScreen()"

ここでは，読んで字のごとく，ゲーム名などをタイトル画面として描画している。スペースキーかリターンキーを押すと，メイン画面関数が呼び出されて，ゲーム開始となる。

・メイン画面関数，"MainScreen()"

いよいよメイン処理部分である。ポイントは，ループと構造体である。この構造体の配列とループを使用することによって，2プレイヤー分の各処理（キー入力，パドルの描画，移動，当たり判定など）を簡潔に記述できる。

もうひとつ重要な部分は，パドルとボールの当たり判定のところだ。壁と同じように，パドルに当たったときに，単純にボール移動座標の入っている変数の符号を反転させるだけでは，45度の角度にしかボールは移動できない。そこでどうするかというと，あらかじめ飛ぶボールの方向を配列"PadDir"に用意しておく。そして，パドルの当たった場所に応じて，その配列を参照してボールの移動方向を決めてやればいい。こうすることによって，パドルの端に行けば行くほど，ボールの反射角が浅くなり，プレイヤーの操作によって，任意の場所にボールを飛ばしてやることが可能になる。あと，コンピュータ対戦時の移動処理だが，ここでは単純に一定の距離に近づいたら，移動を開始するというものにした。

・初期化関数，"InitFunc()"

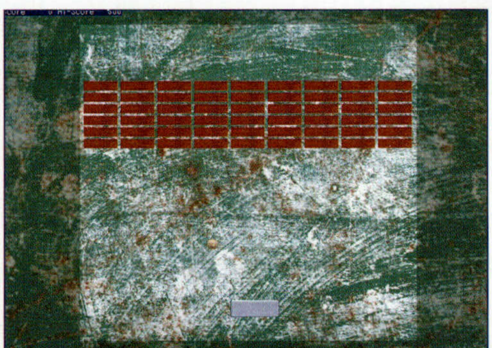
ゲーム情報の初期化を行っている。このなかでは，ゲームセットかどうかも判断していて，その場合は，ゲームオーバー処理関数を呼び出すフラグ"GameOver"を立てている。

・ゲームオーバー関数，"OverFunc()"

前述のゲームオーバー用のフラグが立っているときに呼び出される関数で，ゲームを終了させる一連の処理をして，再びタイトル画面へと戻る。

・点滅関数，"BlinkFunc()"

この関数の引数で，各プレイヤーのうちどのスコアを点滅させるかを決定し，そのスコアを点滅させるために必要な処理をしている。



画面3　シンプルなブロック崩し

以上が大まかなプログラムの解説だが，もちろんこのプログラムを一言一句すべて理解する必要はない（あまりスマートにソースを書けた自信もないし……）。わかるところだけを自分なりに吸収して，ゲームを作るときの参考にでもしてほしい。

●おまけゲーム"Block Buster"

今回のゲームのおまけとして，ブロック崩しをつけてみた（画面3）。これは，本当になんのアレンジも施していない「そのまんま」のブロック崩しだ。マウスを左クリックするとゲームが始まり，マウスでパドルを左右に操作して，ボールを跳ね返し，ブロックをひとつ残らず破壊するとゲームクリアだ。

今回の記事の応用問題として，ブロックの種類を増やしたり，特殊アイテムを加えてみたりと，自分なりにガシガシ改造してもらいたい。

ちなみに，テーブルテニスで使用したグラフィックをそのまま流用しているので，画面的にちょっとアンバランスなものになってしまったが，そこのところはどうかお許し願いたい。*2

冒頭のほうでもなにかゴチャゴチャ書いたが，要するにゲームを作りたいという情熱さえあれば，高度な技術を用いなくとも面白いものができる可能性があるということがいいたかったのだ。

とにかくなにかを作ってみること。プログラミングを学ぶのには実践がいちばんだ。そしてゲームでもなんでも，ほかの人の作ったプログラムを参考に，いろいろと学んでいくことによって，最初はなんだか理解できなくても，だんだんとわかってくることだろう。

ゲームができたらぜひOh!Xに投稿してほしい。そして雑誌という媒体における「作り手」と「受け手」を隔てている，「ミゾ」をできる限り埋めてみんなで楽しくやろう。……これがOh!Xにおけるグルーヴだと僕は（勝手に）考えている。



＊2　今回ちょっと予想していたよりも，もろもろのタスク（連載と特集）が重かったので，連打のあとの休憩（？）ということで……。僕のことをカスタネットライターとでも呼んでくれ（反省）。

Tcl/Tkによる ミニミニゲーム集Part 2

広井誠　Hiroi Makoto

スクリプト言語システムTcl/Tkを使ったミニゲーム集第2弾です。Tclによる手軽な記述とTkによる強力なGUIサポート，そしてプラットフォームを問わない汎用性でゲームシステムとしても有望です。GUI環境でのゲーム作りに挑戦してみましょう。

●はじめに

最近，Windows上で動作するスクリプト言語が注目を集めているようです。以前からCGI（Common Gateway Interface）プログラムで利用されているPerlは別格として，ほかのパソコン雑誌でTcl/Tkを見かけることも多くなっています。ですが，Tcl/Tkでゲームを作ろう，という記事はあまりないようです。趣味でプログラムを作るのであれば，やっぱりゲームを作ったほうが面白いでしょう。そこで，1999夏号に続いて今回もTcl/Tkでパズルゲームと思考ゲームを作ってみました。

1 コラムス



あの「テトリス」と双璧をなす，落ち物系パズルゲームの元祖です。旧Oh!Xで初めて付録ディスクがついた1990年6月号に掲載されていたYet Another Column（泉大介）を参考にしています。懐かしいですね。Tclの処理速度に不安があったのですが，実際に作ってみるとあっけないほど簡単に動いてくれました。

簡単にルールを説明しましょう。コラムスは，上から落ちてくる3連のタイルを使って，積もったタイルを消していくパズルゲームです。3連のタイルには色がついていて，同じ色のタイルを縦，横，斜めのいずれかに3つ以上並べると，そのタイルを消すことができます。タイルは複数の方向に重複して並んでいてもかまいません。消したあとの空間には，上のタイルが落ちてきます。

落ちてくるタイルの操作はキーボードを使って行います。

2，カーソルキー下，スペース	：高速落下
4，カーソルキー左	：タイルを左に移動
6，カーソルキー右	：タイルを右に移動
5，カーソルキー上	：タイルの順番を入れ替える
S	：ゲームの開始

タイルの色はYet Another Columnでは6種類でしたが，今回は5種類，6種類，7種類のなかから選べるようにしました。それぞれトップテンのスコアを記録することができます。

タイルの落下速度は得点に比例して速くなるのではなく，出現したタイルの個数に比例します。得点は，

(消したタイルの数×連鎖ボーナス)^2

です。連鎖とは，タイルが消えて上から落ちてきたタイルによって，同じ色のタイルが3つ以上並んでタイルが消えることです。連鎖ボーナスは連鎖回数が増えるとともに，

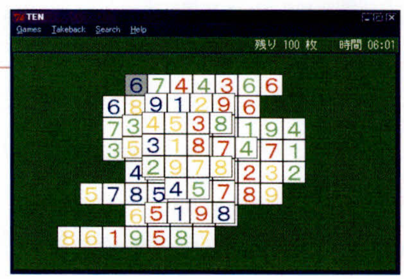
1 → 2 → 4 → 8 → 16 ……

と増加していきます。うまく連鎖を起こして高得点を狙ってください。

・ファイル名

COLUMNS.TCL　：コラムス用 Tcl スクリプトファイル
COLUMNS.TXT　：説明書

2 TEN（テン） 真・上海モード



夏号に掲載されたパズルゲーム「TEN（テン）」のバージョンアップです。TENは，足して10になる2枚のカードを取り除いていくパズルゲームです。カードは赤，青，黄，緑の4種類があり，それぞれ1から9までのカードが4枚ずつ，計144枚のカードをすべて取り除けばクリアです。

TENには，上海モードと四川省モードがありましたが，上海モードのルールが本物の「上海」とはあまりにもかけ離れていたので，「真・上海モード」を作ることにしました。このモードではカードがピラミッド状に積まれていて，次の条件を満たす2つのカードを取り除くことができます。

① 同じ色で足して10になること
② 左右どちらかの辺がほかのカードと接していないこと

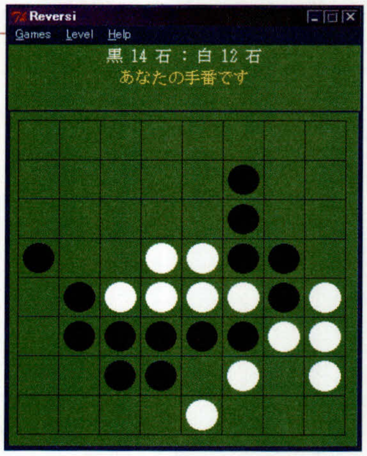
カードはランダムに配置されるため，クリアできる保証はありません。ご注意くださいませ。

・ファイル名

TEN2上海.TCL　：真・上海モード用 Tcl スクリプトファイル

TEN_上海.TCL　：上海モード用 Tcl スクリプトファイル
TEN_四川.TCL　：四川省モード用 Tcl スクリプトファイル
TEN_HELP.TXT　：説明書
TEN_SUB.TCL　：共通 サブルーチン

3 REVERSI（リバーシ）



思考ゲームの定番「リバーシ」です。春号に掲載されたリバーシのバージョンアップ版です。1999年夏に「コンピュータリバーシ対決」が電脳倶楽部で行われました。このとき，拙作のX68000版リバーシが市販のリバーシゲームにけちょんけちょんに叩きのめされたのを見て，メラメラとプログラマ魂が燃え上がり，思考ルーチンの改良に取り組んだのです。その後，電脳倶楽部編集部の鎌田氏も参戦し，三つ巴の熱き戦いが繰り広げられました。この戦いのなかで拙作の思考ルーチンも強くなり，市販のリバーシゲームともいい勝負ができるようになりました。今回のWindows版のリバーシにも，同様の改良が施されていますが，終盤で読み切る手数が長い分だけX68000版より強いかもしれません。初級者レベルの筆者では，Level 1にさえ苦戦してしまいます。もっとも，中級，上級者から見るとまだまだ弱いのでしょう。腕に覚えのある方は，試してみてください。

コンピュータの強さは，レベル0から5までの6段階です。

Level 0　：１手読み，残り8，９手で読み切る
Level 1　：２手読み，残り9，10手で読み切る
Level 2　：３手読み，残り10，11手で読み切る
Level 3　：４手読み，残り11，12手で読み切る
Level 4　：５手読み，残り12，13手で読み切る
Level 5　：６手読み，残り13，14手で読み切る

レベルを上げるとコンピュータは強くなりますが，その分考える時間が長くなります。特に，残り13，14手で最後まで読み切る場合，少々時間がかかることがあります。高速CPUを搭載したパソコンを使ったほうが快適にプレイできます。

・ファイル名
REVERSI.TCL　：リバーシ用 Tcl スクリプトファイル
REVERSI.DLL　：思考ルーチン
REVERSI.TXT　：説明書

4 HEX REVERSI（ヘックスリバーシ）



このゲームは「REVERSI（リバーシ）」のヘックス（六角）バージョンです。ルールは通常のリバーシと同じですが，盤とマスの形が六角形になるため，円盤を挟む方向が6方向と少なくなります。コンピュータの強さはリバーシと同様にレベル0から5までの6段階です。思考ルーチンも，リバーシと同様の改良が施されているので，電脳倶楽部［5］に掲載されたX68000版ヘックスリバーシよりも強くなっています。形を変えただけですが，リバーシとはプレイ感覚がかなり異なるように思います。ぜひ遊んでみてください。

・ファイル名
HEXREV.TCL　：ヘックスリバーシ用Tclスクリプトファイル
HEXREV.DLL　：思考ルーチン
HEXREV.TXT　：説明書

ゲームの実行

今回のゲームはTcl/Tk 8.2を使って作られています。リバーシとヘックスリバーシの思考ルーチン用DLLは，バージョン8.2以前のTcl/Tkではロードすることができません。リバーシとヘックスリバーシを実行する場合，必ずTcl/Tk8.2をインストールしてください。インストール後，Tclスクリプトファイルをダブルクリックするとゲームが実行されます。また，スコアを記録するゲームがあるため，ハードディスクなどの書き込み可能なメディア上で実行してください。

おわりに

このほかにも，春号で掲載された思考ゲーム「カラー」と「アップでポン」のバージョンアップ版を用意しました。思考ルーチンの改良はありませんが，Tcl/Tk 8.2で動作するように修正しました。

ところで，PerlやTcl/Tk以外にもいろいろなスクリプト言語があります。特に日本で開発された，まつもとひろゆき氏のオブジェクト指向スクリプト言語Rubyとonion softwareのBASICに似たスクリプト言語HSPが面白そうです。RubyはTkのインタフェイスを持っているので，簡単にGUIアプリケーションが作れそうですし，HSPはゲームの作成に向いていて，実際にいくつものゲームがHSPで作られています。これらのスクリプト言語はフリーで利用できるので，PerlやTclに馴染めない方は試してみるといいでしょう。自分にあったツールを見つけて，プログラミングを楽しんでください。

参考文献
［1］久野靖『入門 tcl/tk』，アスキー出版局,1997
［2］John K. Ousterhout,西中芳幸 石曾根信 訳『Tcl&Tkツールキット』，ソフトバンク,1995
［3］須栗歩人『Tcl/Tk による Windows GUI プログラミング』，C MAGAZINE 1998年11月号,ソフトバンク
［4］Mark Harrison/Michael McLennan 著，吉川邦夫 訳『Effective Tcl/Tk』，アスキー,1999
［5］広井誠『リバーシ』月刊・電脳倶楽部 Vol.132 満開製作所

Computer Architecture

コンピュータアーキテクチャ

その直感的アプローチ❷

キャッシュのメカニズム

中森 章 *Nakamori Akira*

最近の高クロックCPU搭載コンピュータの性能を左右するのはメモリのアクセス性能だ。高速メモリにも限度がある以上，特にキャッシュ機構の優劣は性能に大きく影響する。今回はそのキャッシュ機構に的を絞って現状のCPUとコンピュータシステムの構成を見直していきたい。

●はじめに

その昔，フォン・ノイマンがプログラム内蔵方式，つまりプログラムもデータと同じようにメモリ中に格納する方式を提唱して以来，その方式は現在のコンピュータアーキテクチャの基本理念となっている。現在の主流のMPUであるPentiumにしろPowerPCにしろ，この方式から脱却してはいない。当然のことながら，ほとんどすべてのMPUは，プログラムの実行に際してメモリへアクセスしなければならない。そして，そのメモリへのアクセス時間がプログラムの実行性能にも影響を与えてしまう。これがフォン・ノイマンボトルネックと呼ばれる現象である。MPUの性能向上のキーポイントのひとつはフォン・ノイマンボトルネックの削減にあるといっても過言ではない。

●キャッシュメモリとは

フォン・ノイマンボトルネックを削減するための手っ取り早い方法は高速（アクセス時間の短い）なメモリを使用することである。世の中にはいろいろな種類のメモリ（記憶装置）があり，アクセス時間に応じて図1のようなメモリ階層を形成している。つまり，高速なメモリは高価であるため大容量を使用することは難しい。そこで，キャッシュメモリという構造が用いられる。

キャッシュ（cache）とは「隠し場所，貯蔵所」という意味で，キャッシュメモリとは原則としてプログラムで意識する必要のない高速なメモリのことである。具体的には図2のように，小容量で高速なキャッシュメモリと大容量で低速なメモリを階層構造に組み合わせる。動作としては，低速メモリ（大容量）の内容の一部をキャッシュメモリ（小容量）にコピーしておき，MPUは通常はキャッシュメモリのみをアクセスする。アクセスすべき内容がキャッシュメモリにない場合は，低速なメモリのほうをアクセスする。厳密には低速メモリの内容をキャッシュメモリへコピーし直し，そこをアクセスする。これは，プログラムのメモリアクセスには局所性があるという経験則が原理となっている。

キャッシュメモリは単にキャッシュと呼ばれることが多い。本稿でも以下ではキャッシュと表記する。また，低速メモリからキャッシュへコピーのし直しはリフィル，またはリプレースと呼ばれる。

現在ではMPUにキャッシュが内蔵されることは珍しくない。しかし，LSIの集積密度がそれほど高くなかった10年くらい昔では，SRAMを使用してMPUの外部にキャッシュを構成していた。とはいえ，SRAM自体が非常に高価だったため，本当に性能の必要な大型計算機等でしかキャッシュは採用されていなかった。ところが，現在主流のRISCではキャッシュの存在を前提とし，メモリへのアクセスは，とりあえずキャッシュヒットするものと仮定してアーキテクチャが決定されている。本当にLSI製造技術の進歩には目を見張るものがある。なお，本稿ではMPUに内蔵されている，特に1次キャッシュを念頭に置いている。しかし，記述自体は一般論に近い。

●キャッシュの構成

キャッシュは，高速，（比較的）小容量である点を除けば通常のメモリと変わりはない。アドレス

図1　メモリ階層（アクセス時間による分類）

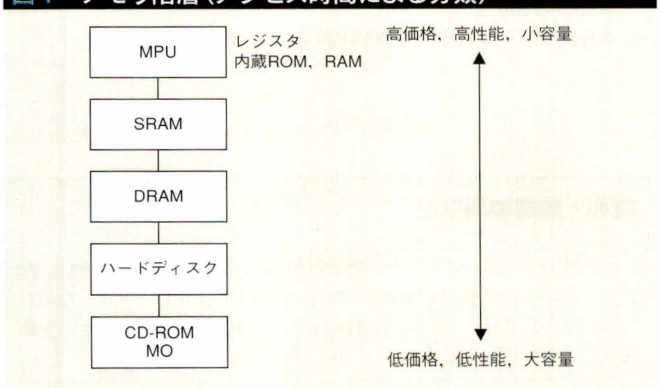
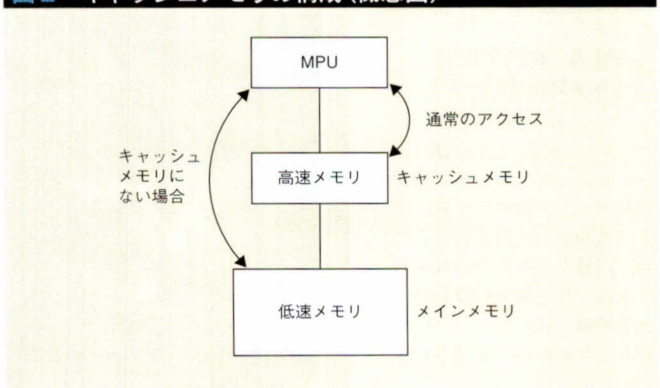


図2　キャッシュメモリの構成（概念図）



を与えると対応するデータが出力される。ただし，低速なメモリ（メインメモリ）の一部をコピーしたものなので，対応するアドレスのデータが格納されていないことがある。これをキャッシュミス（あるいはミスヒット）という。このキャッシュミスを検出するため，特殊な構造を採用している。具体的には，タグ部とデータ部と呼ばれるメモリの組（これをラインまたはエントリと呼ぶ）の集合がキャッシュである（図3）。各アドレスに対して特定のラインが選択され，そのラインのタグ部の内容が与えられたアドレスに一致すればヒットであり，そのラインのデータ部の内容が与えられたアドレスの内容である（有効）ことがわかる。

逆に，タグ部の内容が与えられたアドレスに一致しなければミスであり，データ部の内容は与えられたアドレスのものではない（無効）。現実にはタグ部の中には，ラインの内容が有効なものであるか否かを表すバリッドビットも含まれている。バリッドビットが無効を示していれば，アドレスとタグが一致してもミスとみなされる。

また，データ部の容量はまちまちである。昔は，1ワード（4バイト）の場合が多かったが，現在では4ワード（16バイト）や8ワード（32バイト）が主流である。一般に1ラインのデータ部の容量（バイト数）が大きくなるほど，タグ部に必要なビット数を少なくできる。ただし，データ部の容量を大きくしすぎると，アクセスするアドレス範囲がランダムな場合にキャッシュのヒット率が低下し，性能が低下する。このため，データ部の容量の決定は，予想されるヒット率や利用できる回路規模（この場合は面積）を考慮して決定しなければならない。

●ラインの選択方式（連想方式）

キャッシュではアドレスが与えられると，あるひとつのラインが選択される。この方式には大きく分けて次の3種類がある。

(1)フルアソシアティブ方式
(2)ダイレクトマップ方式
(3)nウェイセットアソシアティブ方式（n≧2）

フルアソシアティブ方式：この方式の概念図を図4に示す。フルアソシアティブ方式においては，与えられたアドレスはすべてのタグ部の内容と比較される。アドレスとタグが一致するラインが存在すればヒット，存在しなければミスである。図4ではライン2がヒットしているので，ライン2のデータ部の内容が有効なデータとして出力される。この方式は，直感的にわかりやすく，ラインをもっとも有効利用できる（したがって，同じライン数ではもっともヒット率が高い）方式であるが，全ラインのタグ部との比較の論理回路が巨大になるため，また，あとで述べる，キャッシュミス時にリフィルするラインを決定するためのLRU（Last Recently Used）処理が複雑になるため，あまり採用されない。

もっとも，LRU処理をあきらめて，FIFO（First In First Out）制御やランダムな選択でリフィルするラインを決定することも考えられる。その場合，ネックとなるのはタグ部の比較論理の回路規模だけである。ライン数が少数（64程度）であれば，連想メモリなどを用いて比較回路を構成することは難しくない。そのため，この方式は，MMUのTLB（Translation Lookaside Buffer）において，仮想アドレスから対応する物理アドレスを選択する（アドレス変換）場合で採用されることが多い。

ダイレクトマップ方式：この方式の概念図を図5に示す。この方式では，与えられたアドレスをデコードして特定のひとつのラインに対応させ

図3　キャッシュの内部構成

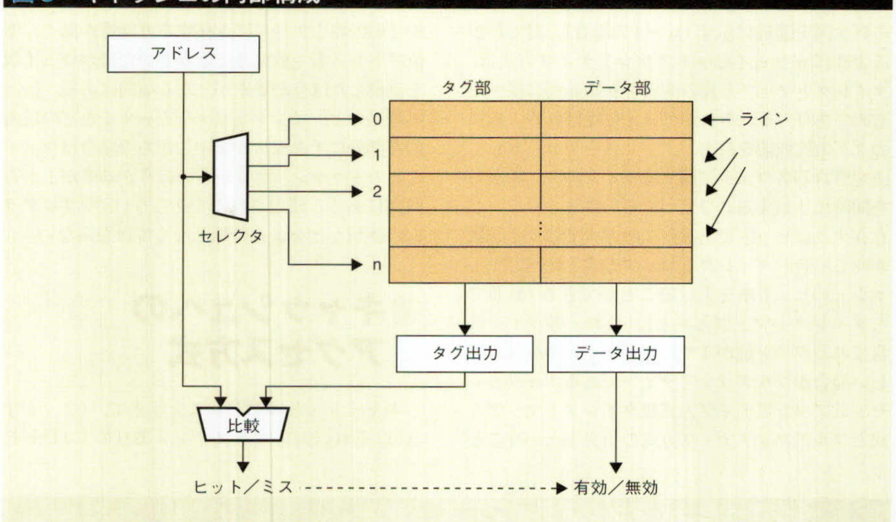


図4　フルアソシアティブ方式

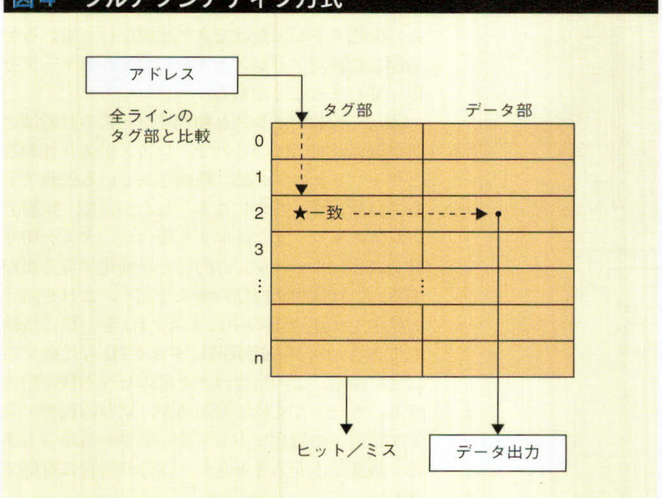
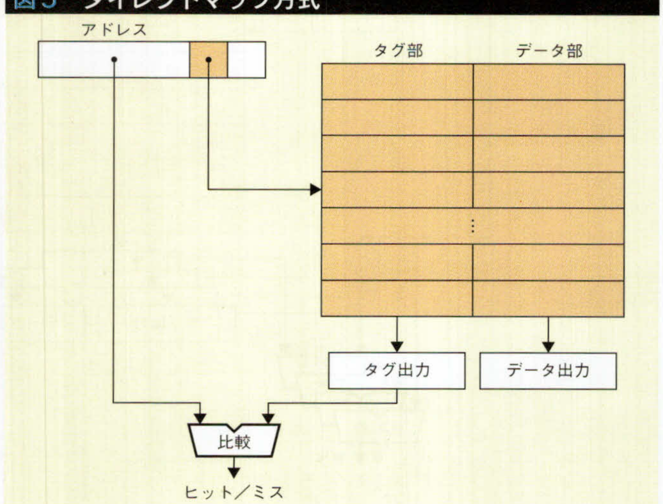


図5　ダイレクトマップ方式



る。デコード，といっても大袈裟なものではなく，単にアドレスの1部分のビット列でラインを選択することが多い。キャッシュの構成が256個のラインからなり，1ラインのデータ部が4ワード（16バイト）だとすれば，現在のMPUではバイトごとにアドレスが割り振られているので，アドレスのビット4からビット11の8ビットでラインの番号を決定すればよい（8ビットなので256種類の値を指定できる）。

もっとも，アドレス内の連続する8ビットで指定した場合，アクセスするアドレス範囲が大きい場合はヒット率が低下するおそれもあるので，アドレスの上位の数ビットも考慮したり，アドレスの2つの部分のビット列の排他的論理和を計算してラインを決定する場合もある。この方式は，キャッシュリフィル時のラインが一意に決定されるのでLRU制御を行う必要がなく，回路構成も単純なため（したがって高速に動作するし消費電力も少ない），1世代前のMPUの内蔵キャッシュに多用されていた。

nウェイセットアソシアティブ方式：この方式の概念図を図6に示す（n＝4の場合）。見てわかるようにnウェイセットアソシアティブ方式は，ダイレクトマップ方式の構成をn個並列に並べたものであり，それぞれがウェイと呼ばれる。n個のタグの比較器を持ち，アドレスをデコードして決定される各ウェイに属するラインのタグ部出力を同時に比較する。ひとつでも一致するラインが存在すればヒットである。この方式ではライン数が同じ場合，ダイレクトマップ方式と比べてキャッシュのヒット率を上げることができる（最悪でもダイレクトマップと同じ）。なお，各ウェイに含まれるライン数が1で，nがラインの総数に等しい場合がフルアソシアティブである。nウェイセットアソシアティブ方式はダイレクトマップ方式とフルアソシアティブ方式の折衷案ということもできる。そこそこの回路規模でそこそこのヒット率が達成できる。最新のMPUではn＝2またはn＝4で構成されることが多い。n＝8というのもあったかもしれない。

ただし，キャッシュライン数が多いことがキャッシュ効率と直接には結びつかないことにも注意したい。同容量のキャッシュサイズの場合，連続的にキャッシュできるエリア，ないしはウェイごとのキャッシュ容量は，

キャッシュ容量/n

で表される。たとえば，8ウェイセットアソシアティブの場合だと，キャッシュがヒットした場合でも，1ウェイのヒットした内容と7ウェイ分のミスヒットの内容を同時にキャッシングしているということになる。キャッシュメモリサイズが限定される場合，さまざまなシミュレーションを行ってもっとも効率がよいと考えられるものにウェイ数は決定される。マルチスレッドで動作するプログラムをキャッシュする場合はプログラムがメモリの広範なエリアに点在する可能性が高く，下位アドレスも一致することが多いためかウェイ数を重視したほうが効率が上がる傾向にある。Java処理系など，インタプリタやカーネルなどの局所的な部分にアクセスが集中しがちな場合はウェイごとのキャッシュ容量が多いほうが効率が上がる傾向にある。使用されるであろうOSやプログラムの種類などをよく吟味しなくてはならない。

キャッシュへのアクセス方式

キャッシュとはアドレスを与えて，（ヒットすれば）それに対応するメインメモリの（コピーしている）データを得ることができるものである。この場合，与えるアドレスが仮想アドレスであるか物理アドレスであるかによって特徴に若干の違いがある。

(1)物理インデックス，物理タグ

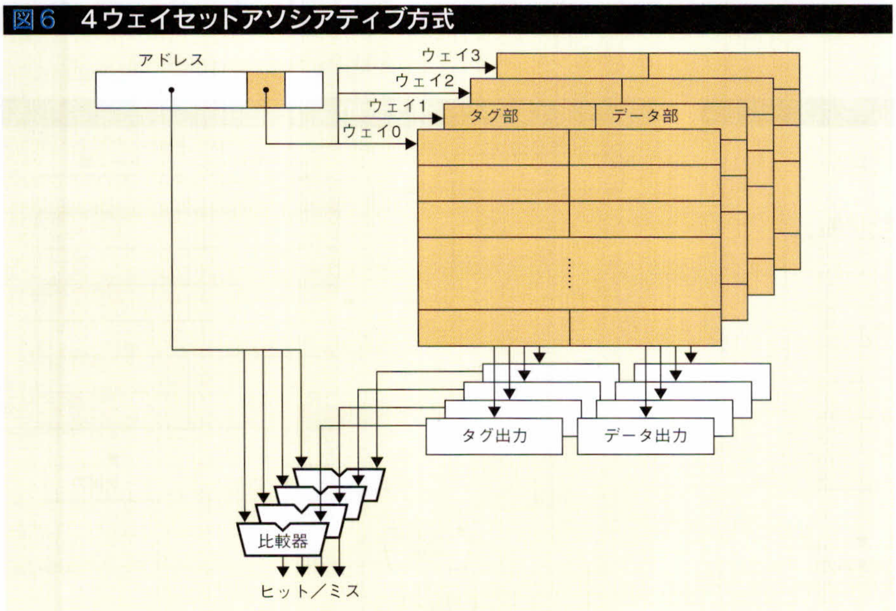
この方式は，一般に物理アドレスキャッシュと呼ばれる。物理アドレスからキャッシュのラインを決定し，出力されるタグ部には物理アドレスが格納されているものとして比較する。キャッシュをMPUの外部に取り付けるしかなかった昔では，MPUの外部バスから出力されるアドレス（もちろん物理アドレス）でキャッシュにアクセスするしかないので，当然物理アドレスキャッシュである。次に述べる仮想アドレスキャッシュと違い，タスク切り替えごとにキャッシュを無効化する必要がないので，制御が簡単である。しかし，仮想アドレスから物理アドレスへのアドレス変換が終了しないとキャッシュにアクセスすることができないので，キャッシュのアクセス時間に余裕がなくなり，高速で動作させることが難しいのが欠点である。

(2)仮想インデックス，仮想タグ

この方式は，一般に仮想アドレスキャッシュと呼ばれる。仮想アドレスからキャッシュのラインを決定し，出力されるタグ部には仮想アドレスが格納されているものとして比較する。この方式ではアドレス変換と同時にキャッシュにアクセスできるため，また，キャッシュ自身にタグ比較の論理を取り込むこともできるため，キャッシュアクセスに余裕ができ，高速で動作させることが可能である。しかし，欠点もある。メインメモリへの最終的なアクセスは物理アドレスで行われるので，メインメモリのデータは物理アドレスで一意に区別できる。つまり，物理アドレスが同じなら同じ場所，物理アドレスが異なれば異なる場所を指す。しかし，仮想記憶で動作している場合，仮想アドレスが同じでも，同じ物理アドレスを指し示しているとは限らない（ほとんどの場合，異なる物理アドレス）。ということは，単純に考えると，仮想アドレスだけでタグ比較を行っていると意図した物理アドレスと異なる場所からデータを取ってしまうことがある。

通常，仮想アドレスと物理アドレスの対応はタスク毎に決まっているので，タスクが切り替わるとキャッシュのタグ部に格納されている仮想アドレスは無意味なものになる。したがって，仮想アドレスキャッシュを採用する場合は，タスク切り替えごとにキャッシュの内容を無効化する必要がある。これは制御回路の増大を招く。これを防ぐ方法としてタグ部の中にタスクIDを一緒に格納しておき，タグの比較時にタスクIDも比較することである。この場合はタグ部のビット数が増大する。また，ごく稀ではあるが，異なる仮想アドレスに同一の物理アドレスを対応させる場合もある。仮想アドレスキャッシュはこの場合に対応できない。

図6　4ウェイセットアソシアティブ方式



(3)仮想インデックス，物理タグ

この方式には特に決まった呼称はない（と思う）。仮想アドレスからキャッシュのラインを決定し，出力されるタグ部には物理アドレスが格納されているものとして比較する。これは，物理アドレスキャッシュと仮想アドレスキャッシュの折衷案である。アドレス変換と同時に仮想アドレスでキャッシュにアクセスし，アドレス変換が終了する頃に，キャッシュから出力される物理アドレスとアドレス変換した物理アドレスを比較する。キャッシュのアクセス時間に余裕ができ，タスク切り替え時の無効化も必要ない。この方式はMIPS系のMPUで採用されている。

●リプレースメント方式

キャッシュはヒットすることが前提とはいえ，現実には頻繁にミスが発生する。この場合，キャッシュ内にメインメモリの新しいコピーを持ってくる必要がある。このとき，どのラインに新しいデータを書き込むのかを決定する方法がリプレースメント方式である。ラインが決定すれば，そこに新しいデータをリフィル（リプレース）する。ダイレクトマップ方式の場合は何の考慮も必要ない。アドレスに対して対象ラインはひとつしかないので，そこをリフィルする。nウェイセットアソシアティブの場合は，与えられたアドレスに対して対象ラインはn個あるので，それからひとつを選択しなければならない。フルアソシアティブの場合はすべてのラインがリフィルの対象である。

(1)LRU（Least Recently Used）方式

この方式はプログラムの（時間的な）局所性という経験則によっている。すなわち，いちばん昔にアクセスされたラインはこれからアクセスされる確率が低いのでそこを更新する，というもっとも妥当な方式である。この方法では，nウェイセットアソシアティブ方式の場合は，各ウェイの同一インデックスにあるn個のラインのアクセス頻度の履歴を記憶しておく。そのために，n＝2の場合は1ビット，n＝4の場合は6ビット，n＝8の場合は22ビット（あってるかな？）のメモリが必要である。フルアソシアティブの場合は全ラインのアクセス頻度の履歴を記憶しなければならないので，ほとんど非現実的なビット数のメモリが必要である。このため，LRU方式は，主としてnウェイセットアソシアティブ方式で用いられる。この方式の欠点としては，ラインへのアクセス（ヒット）ごとにLRUメモリを更新しなければならないので，タイミング的に厳しいということくらいであろうか（あまり問題でもない気はするが）。

(2)FIFO（First In First Out）方式

この方式は，nウェイセットアソシアティブ方式において，0，1，2，……，nの順にリフィルするラインを決定するものである。キャッシュラインがすべて無効な状態からリフィルを続けて行くと，ウェイは，0，1，2，……，nの順にリフィルされていくので，この順に古いデータが格納されているとみなして，その順序で新しいラインを決定する方式である。アクセス頻度が無視されているが，一応，古いラインからリフィルしていくという方針である。ヒットする場合に順序の更新が行われないので，当然LRU方式よりもヒット率は悪くなる。履歴の記憶に必要なメモリのビット数はn＝2の場合は1ビット，n＝4の場合は2ビット，n＝8の場合は3ビットで足りる。LRU方式に比べて少ないビット数で済むのが特徴である。

フルアソシアティブ方式の場合はラインの番号順にリフィルしていけばよいだろう。ところで，インテルのStrongARM（SA-1100）などは，32ウェイセットアソシアティブという（嘘のような）キャッシュ構成を採っているが，さすがにLRU方式ではなく，このFIFO方式か次のランダム方式と推測される。そのせいか，性能はあまりよくない。

(3)ランダム方式

この方式はランダム（無作為）にリフィル対象のラインを決定する方式である。どのアドレスも同じような頻度でアクセスされるはずという予測に基づいている。ラインを指定するために必要なメモリのビット数はFIFO方式の場合と同じである。1クロックごとにそのメモリを更新（たとえば+1）しておいて，リフィルが必要になった場合に，そのメモリの値が（たまたま）示しているラインをリフィルする。ヒット率としてはFIFO方式と大差ない気がする。論理が単純なためか，この方式は結構多くのMPUで採用されているみたいである。マニュアルにLRU方式と明記されていない場合は，おそらくランダム方式であると思って間違いない。

●書き込み制御

キャッシュはなにもリードするだけではない。書き込みを行う場合もある。キャッシュとはメインメモリの内容をコピーしているものだから，常にメインメモリの内容と整合性（コヒーレンシ）が保たれている必要がある。それを実現するためにいくつかの制御方式が考案されている。

(1)ライトスルー（ストアスルー）方式

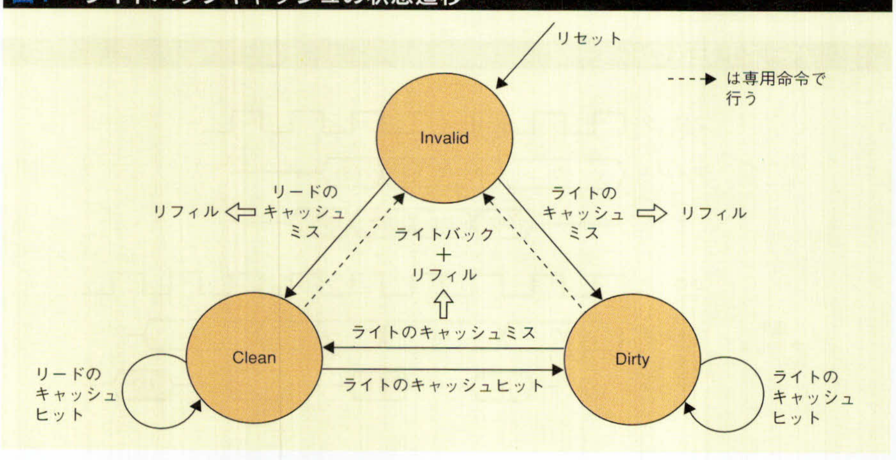
これはライトデータに関して，常にメインメモリにも書き込みを行う方式である。誰もが考えつく方式であろう。ライトアドレスがキャッシュにヒットする場合は，ライトデータをメインメモリと同時にキャッシュのデータ部にも書き込む。キャッシュミスの場合はキャッシュは無視してメインメモリのみにデータを書き込む方式が一般的である。

キャッシュミスの場合には，まずリフィルを行い，そのラインとメインメモリの両方にデータを書き込む方式もある。これは，ライトアロケートと呼ばれる。スタックなど，ライトしたアドレスは再びリードする傾向があるので，あらかじめそこのアドレスをキャッシュに入れておこうという発想である。最終的なリード回数とライト回数は同じはずなので，ライトアロケートが効果的かどうかという点については疑問が残るが。ライトスルー方式を採用する場合，ライトごとにメインメモリへの書き込みバスサイクルが発生するので，連続してライトを行う場合は，前の書き込みバスサイクルが終了するまで次の書き込みバスサイクルを開始できない。このときMPUのパイプライン処理が待ち合わせのために停止してしまう。それを防ぐために，ライトスルー方式を採用するMPUではライトバッファを数段持っていることが多い。逆に，ライトバッファがないと性能が低下する。

(2)ライトバック（コピーバック）方式

この方式はメインメモリへのライトアクセスを

図7　ライトバックキャッシュの状態遷移



最小限に抑える方式である。つまり，ライトが発生しても（ヒットする場合は）キャッシュのデータ部のみしか更新しない。当然，メインメモリとの整合性は保たれなくなる。その代わり，そのラインが整合性が保たれていないことを記憶しておく。そして，あとでメインメモリに書き戻す。そのタイミングは，そのラインがキャッシュにミスし，新しいデータをリフィルしなければならないときである。ライトのいくつか（大半？）はキャッシュにヒットするので，メインメモリに対する書き込みバスサイクルの回数を削減することができる。このメインメモリへの書き込み動作を特別にライトバックと呼ぶ。

ライトバック方式のキャッシュはライトアロケートである。キャッシュミスが発生すると，まずリフィルを行って，そのラインのデータ部にライトデータを書き込む。このとき，メインメモリには書き込まない。また，ライトバック方式のキャッシュでは各ラインが現在のキャッシュ状態というものを持っている。メインメモリと整合性が保たれている状態をクリーン（Clean），整合性が保たれていない状態をダーティ（Dirty）という。この状態を示す情報はタグ部に格納されている。図7にライトバック方式のキャッシュの状態遷移を示す。

●リフィルサイズ

キャッシュミスが発生すると，そのラインはリフィルされる。リフィルは通常ライン単位に行われる。たとえば，ラインのデータ部が4ワード（16バイト）なら，一度に4ワードのデータをメインメモリから読み込む。これは，いったんアクセスしたアドレスの近傍を再びアクセスする確率が高いという，またもやプログラムの局所性によっている。

また，キャッシュのリフィル時に発生するバスサイクルは一般にバースト転送と呼ばれるバスサイクルである。これは，メモリをバスクロック同期で連続的にアクセスする。最近のメモリデバイスはRAMにしろROMにしろページモードというモードを持っている（いま流行のSDRAMも似たような動作をする）。このモードにおいては，最初のアクセスのアクセス時間はやや遅い（というか通常の速さである）が，連続するアドレスの2回目以降は，最初の半分程度のアクセス時間でアクセスできてしまう。4ワードのデータを4回に分けてリードするよりも，4ワードのバースト転送を行った方が遥かに高速なのである（図8）。MPUによっては複数のラインを同時にリフィルするものもある。これは，アクセスする可能性が高いアドレス範囲をあらかじめキャッシュに入れておくほうがヒット率の向上が見込めるためであるが，ページモードとの相性のよさも考慮されているはずである。

現在のMPUでは1回のリフィル時にリードするデータ量（ラインのワード数，または，その倍数）は8ワードが多いようである。MPUによっては32ワード程度まで設定可能なものもある。プログラムの性質（同じアドレスをアクセスする確率の大小）を考慮しながら，最適な値をユーザーが設定できる。

●ノンブロッキングキャッシュとプリフェッチ

キャッシュの構成自体には直接関係はないが，ここでノンブロッキングキャッシュとプリフェッチに関して簡単に説明しておこう。通常，キャッシュミスが発生すると，リフィル動作（バースト転送）が終了するまでパイプラインが止まってしまう。ノンブロッキングキャッシュとは，キャッシュミスが発生してもパイプラインを停止せずに先に進める技術である。

具体的には，リフィルデータを格納するためのリードバッファを何組か用意しておき，キャッシュミスが発生するとリードバッファとリード（またはストア）を発生する命令を関連づける。リフィルはリードバッファに対して行い，キャッシュは暇を見て更新する。その間パイプラインを止めるようなことはしない。リードしたデータをすぐに参照しない場合に効果がある。プリフェッチとは特定の命令（プリフェッチ命令）を実行することで，パイプラインを止めることなく，キャッシュへのリフィルを強制的に行う。同時にキャッシュへのアクセスが発生するリード命令やライト命令を実行しない限りパイプラインを止める必要はない。

個人的にはノンブロッキングキャッシュの効果については懐疑的なのだが，某研究所のシミュレーションによれば非常に有効という結果が出ているので，もしかしたらそうなのかもしれない。プリフェッチは有効に使えばかなり効果がありそうである。

●フェッチバイパス

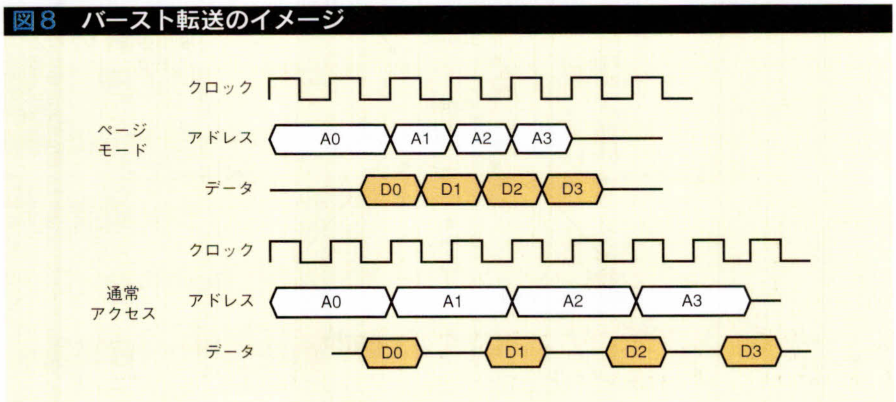
多くのMPUはメモリアクセスがキャッシュにヒットすることを前提に設計されている。ノンブロッキングキャッシュは別であるが，キャッシュミスが発生するとリフィルが完了するまでパイプラインが停止する。命令の連続実行でいうと，止まっては，その後最高速で動き，また止まっては，その後また最高速で動く，という動作を繰り返しているというイメージであろうか。そこで，誰もが思いつくのが止まっている時間がもったいないので，リフィルしているデータをキャッシュに書き込むと同時にMPUにも渡してしまうという方式である。そうするとリフィル中もパイプラインが動作できる。ただし，その間は，命令の実行スピードはバスクロック程度になってしまう。これがフェッチバイパスである。パイプラインクロックとバスクロックに差がありすぎる場合は効果はあまり期待できないが，差がほとんどない場合は非常に有効である。MIPSでは，R3000の命令実行においてこの方式を採用しており，命令ストリーミングと呼んでいる。

●キャッシュロック

高速にアクセスできる作業領域をMPUのチップ内に持ちたい場合がある。あるいは性能にクリティカルな命令領域を常に高速にアクセスしたい場合がある。このような機能を，キャッシュを用いて実現することができる（もちろん，そういう機能が用意されていれば）。キャッシュの特定のラインをキャッシュミスが生じても新たなリフィルによって更新しなければよい。この機能をキャッシュロックという。

キャッシュロックの実装方法はさまざまである。現在のキャッシュの構成方式はnウェイセットアソシアティブが主流なので，それを念頭に置いて話をするが，タグ部の中にロックビットを設けてライン単位にキャッシュロックを指定する方法のほかに，特定のウェイをすべてロックしてしまう方法もある。キャッシュロックの目的は，ある一定量の領域をプログラムの作業領域として確保することであるから，せいぜい4～8ワードのライン単位に細々とロックの指定をしなくてもひとつのウェイに属する1000ワード（4Kバイト）以上の単位でロックすれば十分である。ダイレクトマップ方式のキャッシュでキャッシュロックを採用するMPUもあるが，さすがにこの場合はウ

図8　バースト転送のイメージ



エイ単位でロックを指定することは非現実的であろう。もともとウェイがひとつしかないので，キャッシュがまったく機能しなくなってしまう。

●命令キャッシュとデータキャッシュ

図9にR3000のパイプライン動作を示す。この図で，［ICache］が命令キャッシュへのアクセスを示し，［DCache］がデータキャッシュへのアクセスを示している。図を見ると命令1のデータキャッシュのアクセスと命令3，4の命令キャッシュのアクセスのタイミングが重なっている。命令キャッシュのアクセスは毎回発生するが，データキャッシュのアクセスはロード/ストア命令のみで発生するので，アクセスが重なることは多くないが，まったくないとはいえない。この場合，同じキャッシュからデータを参照することは（どちらかのアクセスを待ち合わせてパイプラインを一時停止しなければ）不可能である。R3000はパイプラインをできるだけ停止しないことを信条としているので，命令キャッシュとデータキャッシュを分けて独立なアクセスが可能にしている。このように，命令キャッシュとデータキャッシュをアクセスする経路を別々に設けるアーキテクチャをハーバードアーキテクチャという。多分，ハーバード大学で初めて提唱されたのでこの名称があるのだろう。MPUではモトローラのMC68020辺りで初めて採用されたように思う。

ハーバードアーキテクチャの欠点（？）は，命令キャッシュとデータキャッシュが別なので命令書き換えに対応できないことである。また，同じアドレスの内容を命令とデータキャッシュでそれぞれ独立に記憶する場合があるので，メモリの無駄と言えば無駄である。逆に，命令とデータを同じキャッシュを持つのがユニファイドキャッシュである。インテルのi486辺りまでがこの方式を採用している。命令書き換えに対応できる（パイプライン動作をしているので，書き換えを行ってからキャッシュに反映されて命令フェッチできるまでに数命令分の遅れがあるはずであるが）し，メモリも無駄にならない。i486はハードウェアアーキテクチャこそRISCであるが，命令セットアーキテクチャはバリバリのCISCなので，メモリアクセスが非常に多い。命令キャッシュとデータキャッシュの同時アクセスによるパイプライン停止が頻繁に発生していると思われるのだが，どのように対応しているのだろう（インテル系のMPUには疎いもので）。レジスタリネーミングにより，レジスタの数がプログラムで記述するよりも多く見えるので，メモリ（キャッシュ）アクセスの頻度がそれなりに少なくなっているのかもしれない（ほとんど想像の域）。インテルもPentium以降は命令キャッシュとデータキャッシュを分離した。命令キャッシュとデータキャッシュのアクセスの競合をなくすためという。ただし，命令書き換えは依然としてサポートしているようであるが（制御が複雑そう）。

●1次キャッシュと2次キャッシュ

図1で示したメモリ階層がMPUの内蔵キャッシュにも当てはまる。チップに内蔵できるキャッシュ（1次キャッシュ）の容量にはチップサイズからくる上限値がある（64K～128Kバイト程度）ので，少し低速で大容量（128K～4Mバイト程度）のSRAMをキャッシュ（2次キャッシュ）として外付けする構成が考えられる。この場合，外付けという性格上2次キャッシュは物理アドレスキャッシュである。MIPSのR4000/R5000/R10000などはこの構成を採用している。

インテルはPentium Pro以降同じような構成を採っている。こちらの場合はMCM（Multi Chip Module）という技術で1チップに見せかけている。Celeron（Mendocino）やPentiumIII（Coppermine）で初めて，2次キャッシュもCPUコアと同じチップに集積されるようになったようである。第1の理由は性能の向上ということだが，さすがにMCMではパッケージの組み立てが大変だったのではないだろうか。聞くところによると，インテルのMPUでは1次キャッシュも2次キャッシュもアクセス時間に差がないそうだ。そんなことをするくらいなら1次キャッシュの容量を増やしたほうが制御が簡単だと思うのだが（同じ1チップ上に集積されていることだし），インテルとしては1次キャッシュが共通のCPUコアに対して2次キャッシュの容量を変化させた製品をいくつか作り，製品系列にバラエティを持たせようとしているものと推測される。2次キャッシュへのクロック供給がCPUコアと同じものと1/2にしたものがあるが，姑息なバリエーションであるといえる。なお最近では，1次キャッシュ（Primary Cache），2次キャッシュ（Secondary Cache）は，それぞれ，L1キャッシュ（Level 1 Cache），L2キャッシュ（Level 2 Cache）と呼ばれることのほうが多い気がする。

●キャッシュのヒット率に関して

キャッシュの目的はメインメモリが低速な場合，プログラムが意識しなくてもメモリアクセスが高速に行えるということである。つまり，メモリアクセスがキャッシュにヒットしなければ性能は低下する。いくら高性能なキャッシュメモリを使用してもヒット率が低ければ意味をなさない。そこで，キャッシュの構成にはヒット率を向上させるための仕組みが盛り込まれている。これまで述べてきたキャッシュの構成はヒット率を向上させる目的で（試行錯誤の末？）提案されてきたものである。ここで，キャッシュのヒット率を向上させるキーワードを明確にしておこう。

(1)大容量

キャッシュメモリの容量は大きければ大きいほどヒット率は向上する。理想はメインメモリと同じ容量を持つことだが，それが実現可能ならキャッシュなんていう特別の仕組みは不要である。

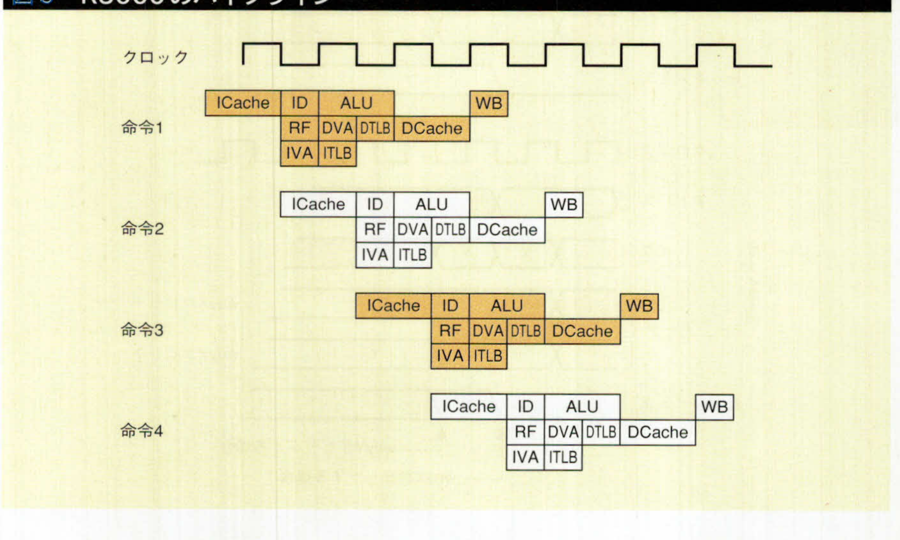
(2)多ウェイ化

フルアソシアティブ方式がもっともヒット率が高い。回路構造上，現実では困難な場合があるので，nウェイセットアソシアティブ構成で，ウェイの数をできるだけ多くするほどヒット率は向上する。

(3)リフィルサイズの増大

プログラムの性質にもよるが，キャッシュのアクセス時間（速い）とバスの転送スピード（遅い）の関係を考えると，一度に（バースト転送で）できるだけ多くのデータをキャッシュ内に取り込むほうがヒット率は向上する。アクセスするアドレス範

図9 R3000のパイプライン



囲が大きくなるほど，多数のラインをリフィルする必要が生じる。そのラインへのアクセス頻度が少なければ，多量のバースト転送がかえってバスネックになるおそれもあるので，リフィルサイズは大きければ大きいほどいいというものでもないが。

(4) LRU 処理

データアクセスの(時間的，空間的)局所性が最大の拠り所である。ラインのリフィル時に，これからもアクセスする可能性の高いラインを書き潰していたのではヒット率は低下する。FIFO，ランダム方式に比べ，LRU方式のほうがヒット率が高い。

●キャッシュ可能領域

MPUによっては周辺デバイスにアクセスするためのI/O命令(IN，OUTなどの命令)が用意されているが，メモリ空間の一部をI/O領域として割り付ける(メモリマップトI/O)ことを前提とし，I/Oアクセスの専用命令がない場合も多い。I/Oデバイスは同じアドレス(I/Oポート)をリードしても同じ値が返ってくるとは限らない。つまり，I/O空間をキャッシュしてはならない。

このほかにも，フレームバッファやDMAの作業領域など，キャッシュ対象にされると具合の悪いメモリ空間もある。このように，同じメモリ空間の中でもキャッシュしてよい(メインメモリのコピーを持ってよい)領域とキャッシュしてはいけない領域が存在する。

これをどのように区別するか。たいていのMPUでは，ある領域がキャッシュ可能であるか否かをMMUで仮想アドレスのページ単位に指定できるようになっている。仮想記憶をサポートしないMPUでは物理アドレスでキャッシュ可能空間とキャッシュ不可能空間が区別されている。MPUの専用端子でキャッシュ可能/不可能を指定する方式もある。すなわち，キャッシュリフィルのバスサイクル中に，ある専用端子をアサート(アクティベート，活性化)することにより，その時点でデータバスに乗っているデータはキャッシュに入れないとする方式である。再び同じアドレスにアクセスする場合は，キャッシュに入っていないのでミスするため，もう一度リフィルが発生する。ここで再び専用端子をアサートすれば，そのデータもキャッシュに入ることなくMPUに渡される。このような仕組みでキャッシュ不可領域を実現できる。もっとも，ライトバック方式のキャッシュでは破綻をきたすかもしれないが。

●バススヌープ

DMAなどメインメモリに直接アクセスする処理を行う場合，メインメモリとキャッシュの内容が食い違う現象が発生する。DMAによるデータはI/Oとは異なり，通常はキャッシュしても構わないデータであるが，食い違いをMPUに通知し，メインメモリとキャッシュの整合性を回復する必要がある。このための一手法がバススヌープである。バスモニタともいう。具体的にはアドレスを指定してそのアドレスにヒットするキャッシュラインを無効化する。この場合，MPUの外部からキャッシュを無効化するアドレス(多くの場合，アドレスバスが使用される)を入力し，専用端子をアサートすることでスヌープが実現される。図10にバススヌープ機能の概念図を示す。しかし，バススヌープ機能を持たないMPUも多い。DMAコントローラは転送の終了時にTC(Terminal Count)割り込みを発生するので，MPUはその割り込みを検知して割り込みを発生し，割り込みハンドラ内でDMAされたアドレスに対応するキャッシュラインを専用命令(キャッシュを内蔵するMPUには大抵用意されている)で無効化すれば事足りるからである。このようにDMAの場合は割り込みによってソフトウェアで処理ができるが，メインメモリを共有するマルチプロセッサ構成ではほかのプロセッサがメインメモリの内容を書き換えたのを検知するのは容易ではない。この場合は，割り込みを使用すると処理

図10　バススヌープの概念図

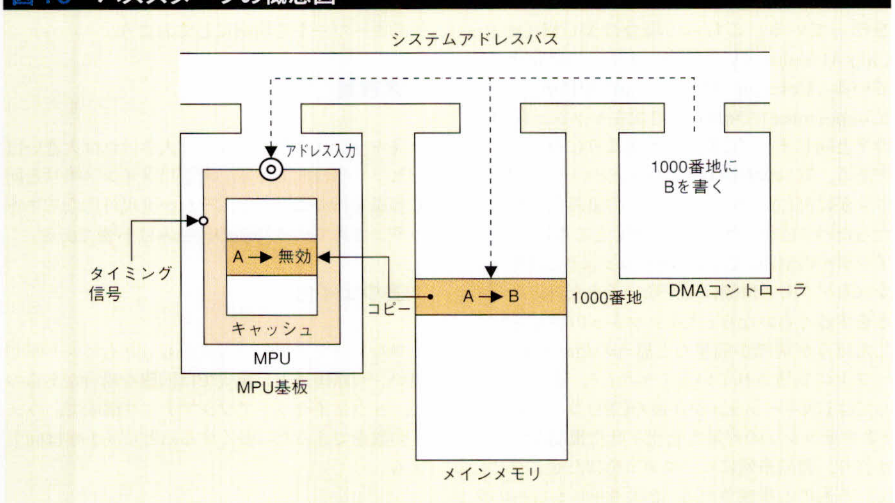
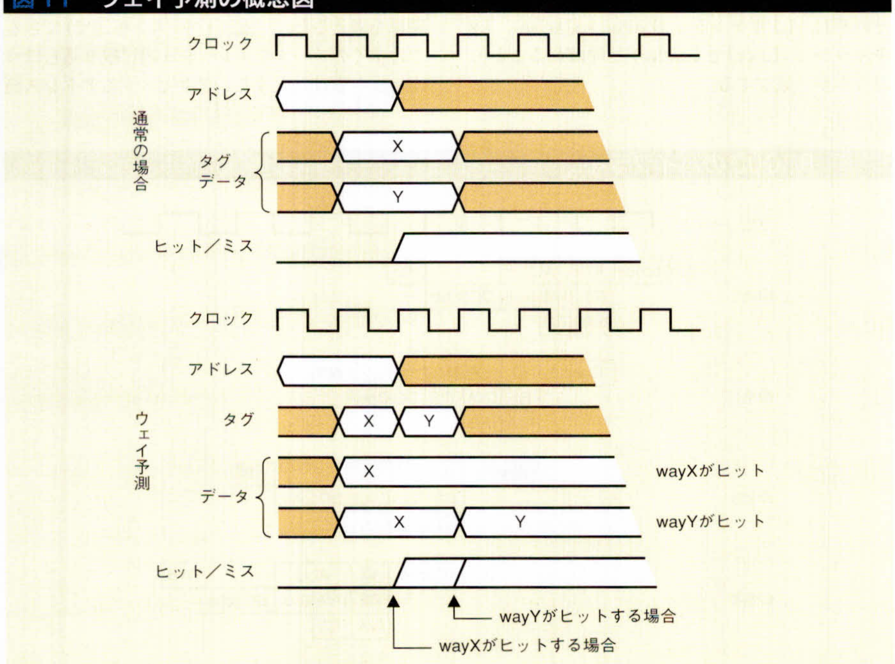


図11　ウェイ予測の概念図



が繁雑になるので，バススヌープが活用される（というか，バススヌープ機能がなくてはマルチプロセッサ対応とはいえない）。

●ウェイ予測

MPUの設計において，キーポイントのひとつが消費電力の削減である。MPUの中でもっとも電力を消費する部分はキャッシュであり，総消費電力の半分程度がキャッシュで消費されているといっても過言ではない。消費電力を削減するために，キャッシュの回路設計においてはメモリセルのブロック分割などの手法が採られることが多い。ウェイ予測もキャッシュの消費電力を削減するために考案された技術である。対象となるのはnウェイセットアソシアティブ構成のキャッシュである。通常の構成では，あるアドレスが与えられたときに，すべてのウェイのタグ部とデータ部の内容を内部バスに出力し，キャッシュヒットするウェイがあればそこのデータを選択する。

いま，ひとつのウェイから1回に出力されるデータが1ワード（32ビット）であるとしよう。このとき，4ウェイセットアソシアティブの場合は4ワードのデータが同時に内部バスに出力され，128ビット分の値が変化する。バスを構成する各ビット線を0から1，または1から0に変化させるためにはトランジスタによって目的の値になるようにビット線を駆動しなければならない。このとき電力を消費する。バス上の値が変化しなければ電力はほとんど消費されない。さて，ウェイ予測とは，内部バスを同時に駆動するのではなく，予測したウェイから順番に駆動していく（キャッシュのヒット/ミスも順番に判定する）方式である。上の例でいえば，1回のバス上の信号の変化は32ビット分のみになり，単純計算で，消費電力は1/4になる。

図11に，2ウェイセットアソシアティブ構成時にウェイ予測を行う場合のタイミングチャートを示す。図でX，Yはウェイのどちらかを表している。どちらのウェイから先にヒット/ミスの判定を行うか（これが予測）については，LRUビットの値から予測する，前回のキャッシュアクセスと同じウェイを見るなどの方法が考えられるが，決定版という方法はないようである。ウェイ予測が当たればヒット/ミスの判定にロスはないが，予測がはずれればヒット/ミスの判定に1クロック程度のロスが生じる。この場合，確かに性能は若干低下するが，性能と消費電力のどちらに重点を置くかで，ウェイ予測の採用/不採用が決まるであろう。

●MC680x0でのキャッシュ構成

MC680x0ではMC68020でキャッシュが内蔵された。ただし，命令キャッシュのみである。その構成を図12に示す。256バイトのダイレクト

図12　MC68020の命令キャッシュ

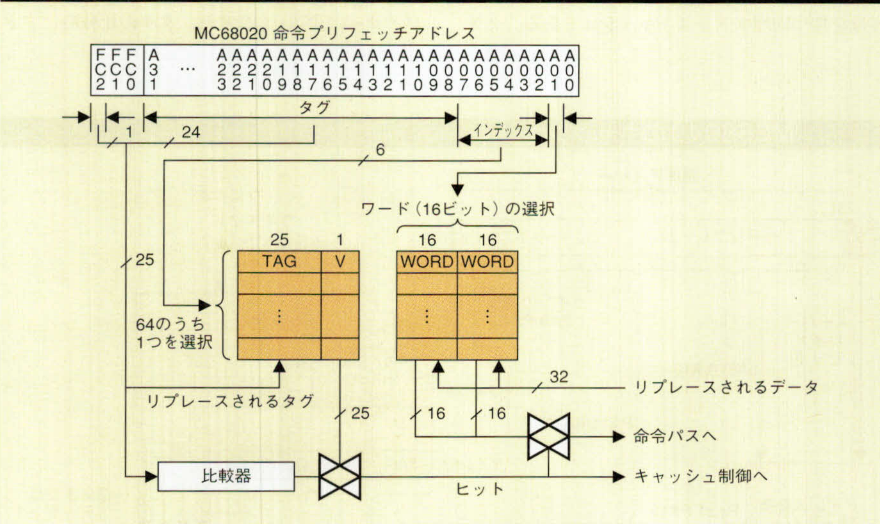


図13　MC68030のデータキャッシュ

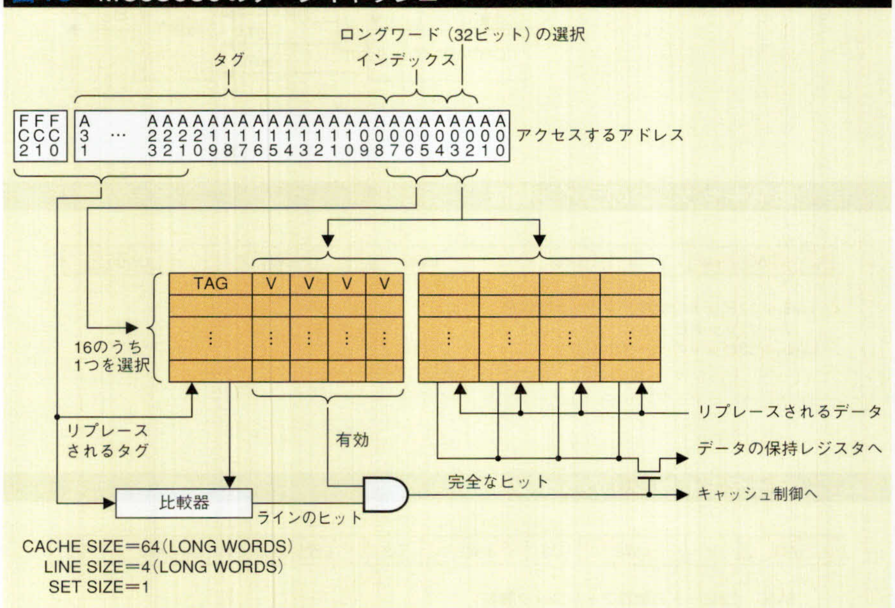
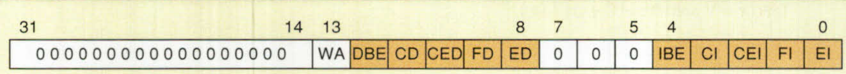


図14　キャッシュ制御レジスタ



WA ＝ライトアロケート
DBE＝データバーストイネーブル
CD ＝データキャッシュのクリア
CED＝データキャッシュ内のエントリのクリア
FD ＝データキャッシュの凍結
ED ＝データキャッシュのイネーブル
IBE ＝命令バーストイネーブル
CI ＝命令キャッシュのクリア
CEI ＝命令キャッシュ内のエントリのクリア
FI ＝命令キャッシュの凍結
EI ＝命令キャッシュのイネーブル

マップ方式で，1ラインは1ワード(4バイト)の容量を持つ物理アドレスキャッシュである。タグ部には機能コードのビット2(ユーザー/スーパーバイザーの表示)も含まれ，タグの比較時にアドレスと同時に比較される。MC680x0では同一の物理アドレスでも機能コードによって物理空間が区別されるからである。MC68030では命令キャッシュに加えて，データキャッシュも内蔵された。図13にMC68030のデータキャッシュの構成を示す。256バイトのダイレクトマップ方式で，1ラインは4ワード(16バイト)の容量を持つ物理アドレスキャッシュである。書き込み制御はライトアロケート可能なライトスルー方式である。1ワードの容量がMC68020に比べ4倍に拡張されている。本来なら1ラインに1ビットあれば十分なバリッドビットがワードごとに用意され全部で4ビットあるのが特徴である。キャッシュのリフィルを1ワード単位でも4ワード単位(バースト転送)でも行えるような設定が可能なためであろう。なお，命令キャッシュもまったく同じ構成をしている。図14にMC68030のキャッシュ制御レジスタ(CACR)を示す。この図を見ればわかるが，キャッシュロック(凍結)も可能である。

MC68040ではキャッシュ構成ががらりと変更された。図15にキャッシュの構成を示す。4Kバイトの4ウェイセットアソシアティブ方式で，1ラインは4ワードの容量を持つ仮想インデックス物理タグキャッシュである。書き込み制御はMMUでページ単位にライトスルー(ライトアロケートはしない)方式とライトバック方式を選択できる。リプレースメント方式はランダムである。図16，図17に命令，データキャッシュのライン構成を示す。バリッドビットはラインに1ビットのみとなった。不思議なのは図17でワード単位にダーティビットが用意されている点である。リフィルやライトバックはライン単位に行う(バリッドビットが1ビットしかないため)のでラインごとに1ビットあれば十分なはずなのだが。図18にMC68040のキャッシュ制御レジスタを示す。それぞれのキャッシュのイネーブル(許可)ビットしかなく，キャッシュロック機能はなくなった模様である。

図15　MC68040のキャッシュ

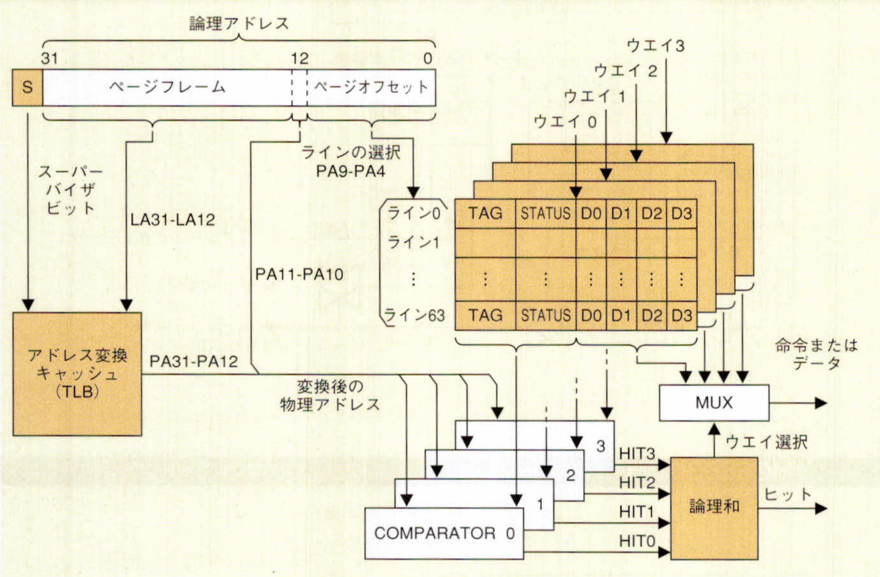


図16　命令キャッシュのライン構成

TAG	V	LW3	LW2	LW1	LW0

TAG ― 22ビットの物理アドレスタグ情報
V ― バリッドビット
LWn ― 32ビットのデータエントリ

図17　データキャッシュのライン構成

TAG	V	LW3	D3	LW2	D2	LW1	D1	LW0	D0

TAG ― 22ビットの物理アドレスタグ情報
V ― バリッドビット
LWn ― 32ビットのデータエントリ
Dn ― Lwnに対応するダーティビット

INVALID＝$\overline{V}$
VALID＝V and $\overline{(D3+D2+D1+D0)}$
DIRTY＝V and (D3＋D2＋D1＋D0)

図18　キャッシュ制御レジスタ

31	30 … 16	15	14 … 0
DE	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	IE	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

DE＝データキャッシュのイネーブル
IE ＝命令キャッシュのイネーブル

●おわりに

主として，MPUに内蔵されているキャッシュの概要を述べてきた。キャッシュの動作を少しでも理解していただければ幸いである。まあ，直感的なアプローチを主眼に置いているので雰囲気だけでも感じていただければよいのではないか。なお，今回はマルチプロセッサ構成時のキャッシュの動作については複雑になるので意図的に省いている。興味ある人は自習してほしい。

ところで，本稿ではnウェイセットアソシアティブのn個のダイレクトマップ形式のキャッシュを指すものとしてウェイという表現を使ってきたが，本来の意味が「n通りのセット」ということを考えるとセットといったほうが正確かもしれない。まあ，ウェイと表現するのは職業病なので勘弁願いたい。さて，次回のテーマは未定である(個人的にはパイプラインをやろうかなと思っているが)。リクエストのある方は愛読者カードでお願いしたい。それでは，また。

Step to the Black Arts

最近のパソコンというものは，非常に多彩なことができる。少なくとも，それだけのシステムを搭載している。1GHzに迫るCPUクロック，大容量キャッシュ，内蔵されたベクトル演算ユニット，超高速メモリを搭載したビデオカード，3Dアクセラレータ，ジオメトリエンジン，物理音源モデル……。

これらの装備を必要とする処理は，普通にパソコンを使っているだけであちらこちらにちりばめられ，さまざまなモジュールで実装された多彩なことを行うために直接に間接に使われている。インターネットの普及で動的コンテンツが当たり前になってきて，サウンドや3Dといった，従来はなくてもなんとかなった部分の比重が次第に大きくなっていく傾向にある。ユーザーインタフェイスも凝ったものになれば，総合的なマシンパワーを必要とするようになってくる。

その割にスーパーコンピュータクラスのパフォーマンスを実感できるようなアプリがないのは相変わらずだが，まもなく現れるであろうDirectX7対応のゲームやDirectX8 &デュアルプロセッサによる新次元CGの境地なども今年中には垣間見ることができるだろう。最先端はいつもゲームというのも相変わらずの状況だ。ゲームに関するプログラミングトピックは特集のほうを見ていただくとして(ユーザーレベルでもかなりのパフォーマンスを引き出せる環境が整いつつあるのは素晴らしいことだ)，それ以外のトピックを探してみよう。Macintoshの Aquaなど，ユーザーインタフェイスの改善という方向でもマシンパワーが使われようとしているのは喜ばしいことだ。かつてのWindowsで同種の試みとして注目されたChrome Effectが凍結となって久しいのだが，時代は確実にそういった方向に向かっているということだろう。このように基本要求スペックが上がれば，マシンパワーも相乗効果で上がっていき，ユーザーが引き出しうるマシンポテンシャルも上がってくるのだ。

最近ではパソコン以外のものが注目を集めだしている。たとえば，PDAであり，携帯電話系機器だ。特に，カラー化，Java搭載の話題などが先行気味なものの携帯電話系機器は次世代パーソナル端末としてもっとも成長の期待されている分野だ。表示能力に制限はあるものの，一応使いものになる程度のものを備え，プログラマブルでさらに高速データ通信への道も示されている。もちろん携帯性のよさと市場の規模が大きいという最大のメリットも見逃せない。新型StrongArmなどの小型超高性能プロセッサも続々開発されており，可能性は計り知れない。

そして時代とともにプログラマ自体に要求されるスキルも変化している。プログラマ受難の時代である。とはいえ，趣味のプログラムという範疇なら，手を出せる範囲，つまり「遊び場」がどんどん広がっていくことは素直に喜んでよいことなのだ。

期待のニューバージョン Future BASIC 3

古籏一浩　KaZuhiro FuRuhata

Macintosh用の手軽な開発環境として知られるFuture BASICですが，今回はプログラミングの話からはちょっと離れて，このFuture BASICの最新バージョンの話題について紹介していきたいと思います。

一時期開発中断という噂もあったFuture BASIC 3ですが，ついに現物が目の前に登場しました。ただし，現状ではリリースバージョンゼロという状態(まあβ版みたいなものです)。この原稿を書いている時点で次々と改良，バージョンアップされている状態です。また，現在存在するのは英語版のみで日本語版は，まだ影も形もありません。

日本語版がいつ出るかは未定ですが，前バージョンと異なり日本語など2バイトにも対応して登場するようです。日本語に関する命令も用意されるようですので，いままで日本語処理に苦労していた部分が楽になるのではないかと思います。

それでは，Future BASIC 3はどのように変わったのか紹介していきたいと思います。なお，日々，命令の追加や削除，仕様変更などが行われていますので，実際の発売版との多少の違いはあらかじめご了承ください。

PowerPC対応

Future BASIC 3で，やっとPowerPC対応になりました。「えっ！」といわれそうな感じですが，Apple社のOS戦略に振り回された結果でもあります。遅れに遅れてやっとPowerPCコンパイラが搭載されました。もちろん従来同様68Kコンパイラも搭載されており，「68Kのみ」「PowerPCのみ」「68K+PowerPC (Fat Binary)」という3種類のオブジェクトを作成することができます[*1]。

どのタイプのアプリケーション(68K，PowerPC，Fat)を作成するかは図1のようにPreferenceメニューで表示される設定ダイアログか，Commandメニューから68K/PPC/Fatを選択するだけです。これだけで68K，PowerPC, 68K+PowerPC (Fat)用のアプリケーションが作成できます。

また，いままではランタイムといって実行に必要なライブラリはひとつで，自分の作成するアプリケーションによって変更することができませんでした。しかしFuture BASIC 3ではゲームなど速度を要求するものには速度優先のランタイム，実用ソフトなど安定性を求め速度を重視しないもの，Future BASIC 2との互換性を重視するもの，オブジェクト指向のランタイムの4つを選択することができるようになりました。

Future BASICはほかのBASICと違ってBASICプログラム内にインラインアセンブラを使用することができます。普通のBASIC，VisualBASICやREALbasicではありえない(できない)ことですが，Future BASICは68Kのアセンブラだけでなく PowerPCのアセンブラも記述することができます。見方によっては危険きわまりない(暴走する可能性も多分にある)のですが，別の見方をすれば「なんでもできる」「Future BASICでFuture BASICが作れる」ということにもなります。ゲームなど速度が要求される部分においてはコンパイラの性能に大きく左右されてしまいます。コンパイラに左右されては困る部分，つまりアセンブラで記述すればマシンの能力を100%発揮することができます。

と書いてみたものの，最近のマシンは非常に高速かつパワフルになりました。インラインアセンブラなんて必要ないじゃないか，という人もいるかもしれません。現在のコンピュータは，ほぼブラックボックスになってしまい，内部でなにが行われているのかを把握することは非常に困難です。ましてやアセンブラなど見ることも言葉として聞くことも少なくなっています。しかし，いくら高度なことを行っても，やろうとしても土台の部分がしっかり把握できていなければ，いつか崩れます。Future BASIC 3は，手軽さと同時に本当に基礎的な部分をも学ぶことができる珍しい開発環境であり言語なのです[*2]。

＊1　REALbasicと違いWindowsコンパイラは存在しません。あくまでもMacintosh用なのです。
＊2　Macintoshだとハードウェア情報が一部しかわからないので基礎から学ぶという点ではイマイチですが……

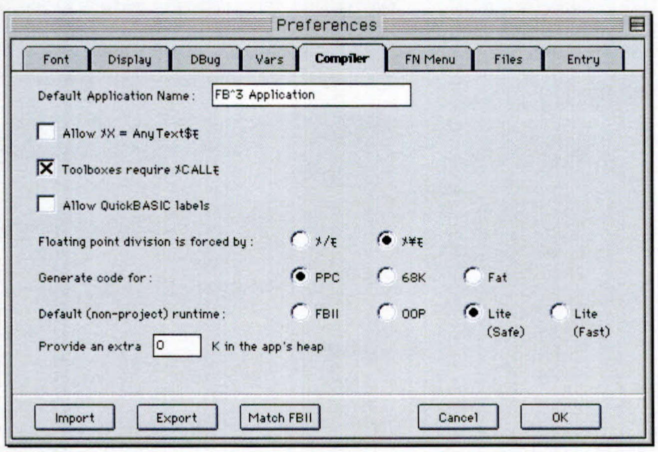
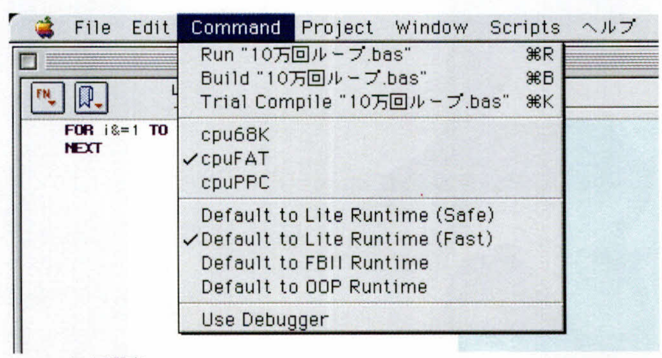


図1　プレファレンスによる設定。Compilerの部分でGenerate code for:で作成する形式を指定する

起動画面/エディタ/コンパイル速度

以前のFuture BASIC 1.0.x, Future BASIC 2では図2のようにスプラッシュウィンドウ(起動画面)が表示されていました。Future BASIC 3ではなにも表示されません。アバウト画面を出すと図3のようになります。格段に格好よくなってます。

Future BASIC 3を起動するといっても実際はエディタを起動すること



メニューによる設定

になります。従来はエディタとコンパイラは一体になっていましたがFuture BASIC 3からはエディタとコンパイラは別々のアプリケーションになりました。だからといって特別に意識することなく従来どおりの方法/コマンドキーでコンパイルすることができます。

エディタとコンパイラ，そして一度アプリケーションを作成してから実行するようになったため，以前のようにプログラムミスでハングアップしてFuture BASICごと道連れでお亡くなりになってしまうことはなくなりました。また以前はFuture BASIC上で実行したときとアプリケーションを作成して実行した場合で実行結果が異なってしまうことがありましたが，これも一度アプリケーションを作成し実行するため実行結果が異なるという事態はなくなっています。

エディタとアプリケーションが分離した影響もありますがコンパイル速度に関しては格段に遅くなっています。以前のバージョンのように，ちょっと作ってはコンパイルという手軽さは少しなくなってしまいました。

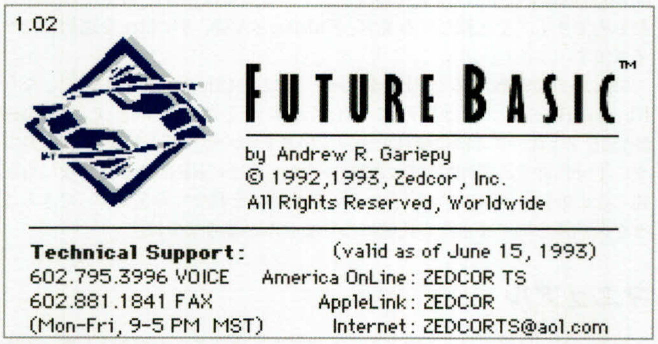


図2　Future BASIC 1.0の起動画面



Future BASIC 2の起動画面



図3　Future BASIC 3のアバウト画面。格好よくなった

コンパイル後に構築されるアプリケーションも特にメニューなどを指定しない場合，従来と異なり自動的にファイル，編集メニューが作成され最低限必要なメニュー項目が追加されます。また，

```
FOR i&=1 to 10000000
NC(a&)
NEXT
```

などのように非常に処理に時間がかかる場合でもイベントや，その他のタスクが処理されているため，以前のように「どうにも止まらない！」という間抜けな事態はなくなりました。ファイルメニューから終了(Quit)を選択すればちゃんとアプリケーションを終了させることができます。

アプリケーションが起動したときも従来はWINDOW OFF命令を使用しないと強制的に1枚ウィンドウが表示されていましたが，必要なときに必要に応じて表示されるようになりました。このためWINDOW OFF命令は意味のないものとなりコンパイルすると警告が表示されます。

コンパイルエラーは従来のようにひとつのエラーだけ表示するのではなく，図4のようにエラー箇所を複数まとめて表示するようになりました。ここで「Click here to view error」をクリックするとエラー箇所に自動的にジャンプしエラー箇所が図5のようにハイライト表示されます。とにかく一度にエラーを表示してくれるようになったのでデバッグ効率がよくなりました。コンパイル速度がもっと速いとよいのですが，まだまだ改良中なので期待したいところです。

命令の追加/削除

Future BASIC 1.0から2のときには命令の変更などは，ほとんどなくコンパイラとエディタの改良，日本語化などでしたがFuture BASIC 2から3へは命令がいくつか追加，削除されています。どの命令が追加/削除され

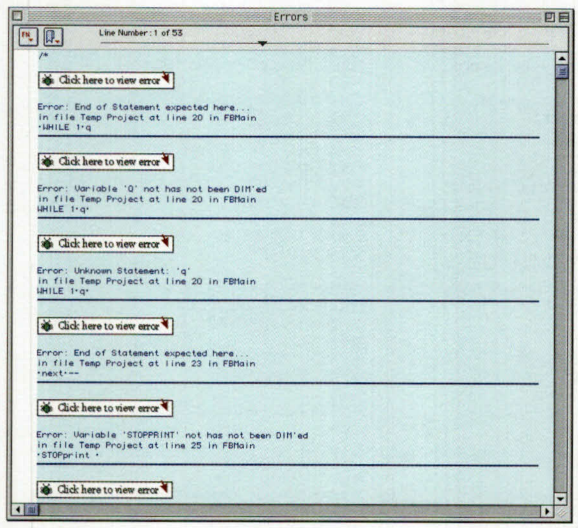


図4　コンパイルエラーはまとめて表示される

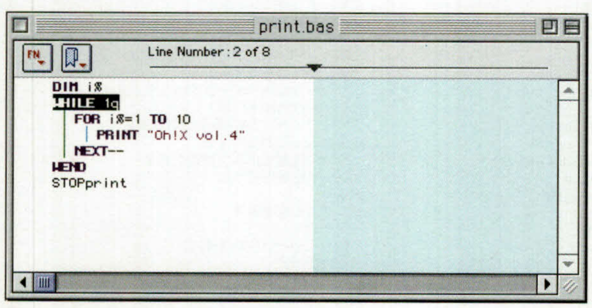


図5　エディタ画面でエラーが確認できる

たかは表1を参照してください。

削除された命令は，ほとんど利用しないか利用できないものばかりです。イベントを一切受け付けないというマシンパワーフル回転は標準状態ではできなくなり，常にイベントを受け付けるようになっています。もちろんマシンパワーをフルに発揮したい場合は一切イベントを取得せず処理させることも可能です（一部の例外を除いて，おすすめできることではありません）。

表1を見ても，あんまり変わっていないという感じもします。グラフィックやサウンドなどが強化されたわけでもなく機能的にはシンプルなままです。もともと複雑な処理はToolboxを呼び出して使うようになっているためでしょう。Toolboxの呼び出し方法ですが基本的には従来と変わりません。また，標準では使えないものはPascal Converterを使って変換していましたがFuture BASIC 3では，

```
TOOLBOX FN HasDepth(LONG, INT, INT, INT) = INT `0xAAA2
```

のようにToolbox名と引数，そしてトラップアドレスを記述して強引（？）に使うことができるようになりました。BASICといいつつ，できることはC言語並みです。手軽にアプリケーションが作れるという点ではREALbasicに負けますが，非常にパワフルであることには違いありません[*3]。

＊3　GUI（グラフィカルユーザーインタフェイス）を手軽に構築するためのProgram Generatorというソフトが Future BASIC 2に標準でついていますが，原稿執筆時点ではFuture BASIC 3版は存在していないので，どうなるかは不明です。

表1　追加された命令群

命令	内容
#DEFINE	新しい型にする
ACOSH関数	
ACOS関数	アークコサイン
ANNUITY関数	
ASINH関数	
ASIN関数	アークサイン
ATANH関数	
ATAN関数	アークタンジェント
BEGIN ENUM～END ENUM	値の割り当て
BEGIN GLOBALS～END GLOBALS	グローバル変数定義
BEGIN RECORD～END RECORDS	レコード定義
BEGINASSEM～ENDASSEM	680x0，PowerPCアセンブラ記述
COMPILE SHUTDOWN	
COMPILERVERSION関数	コンパイラのバージョンを返す
COMPOUND関数	
COSH関数	
DIM SYSTEM	アプリケーションメモリ割り当て
EXIT FOR	FORループから抜ける
INKEY$ <ioChannel>関数	モデム/プリンタポートからの入力
LIBRARY	共有ライブラリ指定
OFFSETOF関数	レコードのオフセット
PROC関数	ラベルアドレスを返す
REGISTER ON/OFF	レジスタ割り当て
SINH関数	
SIZEOF関数	型のサイズ（バイト数）を返す
STRINGLIST	STR#リソースへの保存
TOOLBOX	ツールボックス呼び出し関数定義
TYPEOF関数	型を返す
UNIVERSALFN関数	ルーチンデスクプリタ
UNIVERSALPROC	ルーチンデスクプリタ
命令	内容
VAL&関数	文字列を数値（32bit）に変換する

削除された命令群

命令	内容
BREAK ON/OFF	コマンドキーとピリオドキーの有効/無効スイッチ
CVB関数	BCD小数値に変換
DEF MOUSE	マウスイベント取得指定
DIALOG ON/OFF	ダイアログイベント取得スイッチ
ENTERPROC%	コードリソース定義
ERRMSG$	エラーメッセージを返す
EVENT ON/OFF	イベントのオン/オフを切り換える
EXITPROC%	コードリソース定義の終わり
GET	画面キャプチャー
GET FILE INFO	ファイル情報を返す
GET VOLUME INFO	ボリュームのパラメータブロックを返す
LCOPY	ウィンドウ内容を印刷
MENU ON/OFF	メニューイベント有効/無効切り換え
MKB$	数値をBCDにパックする
MOUSE (non-event)	マウスの位置とボタン情報を返す
OPEN TALK	スピーチ処理を行う
PAGE LPRINT	紙送り
REGISTER	レジスタの値を返す
SGN関数	符号を返す
SPC	指定した数だけ空白を返す
TAB	タブ位置指定
TALK	スピーチ情報指定
USR OPENRFPERM	指定ファイルのリソース番号を取得する
WINDOW OFF	起動時にウィンドウを表示しないようにする

演算

演算に関しては大幅な変更があります。整数演算に関しては従来と変わりませんが，浮動小数点演算に関しては従来とは異なっています。いままではBCDといって演算誤差のない方法で行われていました。しかし，これは非常に低速で全体的な実行速度を低下させてしまう要因のひとつでした。そこでFuture BASIC 3では，ごく普通の演算方式になりました。PowerPCではCPU内に浮動小数点演算ユニットがありますから，劇的に高速化されています。従来は68K＋BCD演算ということで3Dゲームなどは速度面で難しかったのですが，今後は異常な速度低下を気にすることなくゲームや実務アプリケーションを作成することができるようになります。

アプリケーションサイズ

アプリケーションサイズは使用するランタイムや68K，PowerPC，68K+PowerPC（Fat）により異なります。ここではSTOP命令だけ記述したものでサイズを比較してみます。Future BASIC 3はLiteを選択したサイズです。

68KのみでFB1が39K，FB2が45K，FB3が160KとなりFuture BASIC 3では格段にサイズが大きくなっています。Future BASIC 2と比べて4倍に膨れ上がっていますが，それでもわずか160KBです。十分に許せる範囲ではないでしょうか。表3にREALbasicでなにも記述しない最低サイズのアプリケーションサイズを載せておきます。できることが全然違うのでまったく比較になりませんが参考までに。

マニュアル

マニュアルは従来はオンラインヘルプで閲覧することができました。もちろん紙のマニュアルもありました。Future BASIC 3ではマニュアルはすべてオンライン化されeDoc Readerで読むことができます。もちろん印刷も可能です。マニュアルは以下のように分かれています。

FBII Conversion（Future BASIC 2からの変更点）
FB³ Editor（Future BASIC 3のエディタの使い方）
Learning to Program With FB³（プログラム学習）
Program Generator（プログラムジェネレータ）
Project Manager & Debugger（プロジェクトマネージャーとデバッガの使い方）
Reference Manual（リファレンスマニュアル）

これらのマニュアルに関しては現在有志による翻訳作業が進められています。Future BASIC 2のような面白いマニュアルにはならないと思いますが鋭意翻訳中とのことです。

マニュアルは図版なども入っているので割と見やすくテキストのコピー，1ページを画像としてコピーすることもできます。ページの表示速度は一瞬で速度的なストレスはまったくありません。表示の比率も割と自由に指定できるので読みにくいということはありません。

実行速度

さて肝心の実行速度ですが，シンプルなもので計測してみました。単位はtickなので1tick=1/60秒となります。60 ticksで1秒になります。なお，ランタイムはFBII互換のものにして計測しました。

全体的に68Kのランタイムは以前のものより低速になっています。さすがにPowerPCのランタイムはすべて高速になっています。特に浮動小数点演算に関しては68K，PowerPCランタイムとも格段に高速化されています。100万回浮動小数点演算の結果の値ですが，これは間違いではありません。実に1550倍，単純な加算でも60倍ほど速くなっています。

メモリの読み書きも速くなってはいますが1.2倍ほどであまり高速化されていません。劇的な高速化が望めるのは演算で，あとはPowerPCネイティブになったので速くなったといったところでしょう。

Future BASIC 1,2で作成したプログラムをFuture BASIC 3でコンパイルすると99%エラーか警告(ワーニングエラー)になります。高速化の恩恵を受けたい場合は，いくらかプログラムの修正を必要とします。しかし，それを差し引いてもPowerPCネイティブで動作するというのは大きなメリットです。

また，従来と異なりランタイム部分はFuture BASICで作成されているため自分で書き直して使用することができ，サイズを軽減したり独自の機能を追加することが可能になっています。Future BASIC 3をFuture BASIC 3で作っているわけです。

GUIが手軽に構築できるProgram Generatorに関しては，現物がないので不明なままです。REALbasic並みにサクサクと見栄えのよいアプリケーションが作成できるようになるのかもしれません。もっとも，気に入らなければ自分で作成してしまえばよいのです。Future BASIC 3には，それだけのパワーがあるのですから。

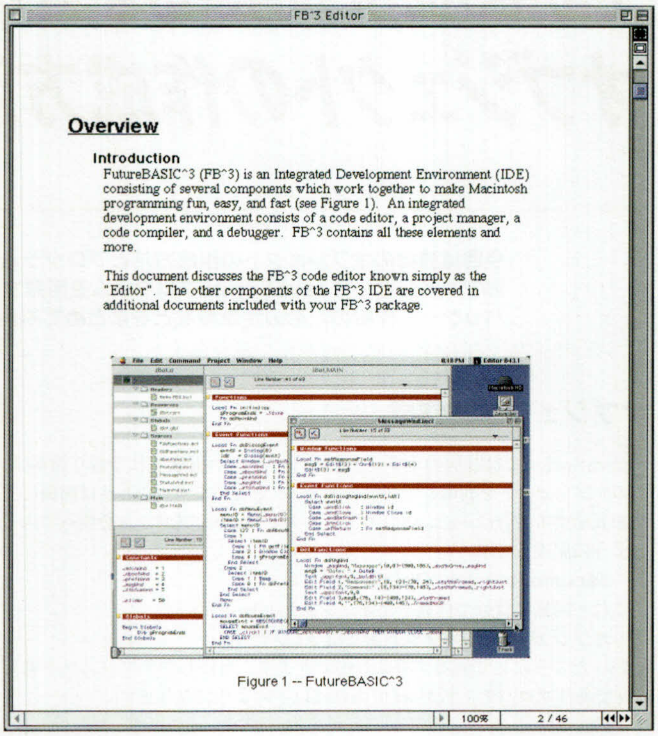


図6　オンラインマニュアル

表2(単位はバイト数)

	FB1	FB2	FB3
68K	39934	45604	160800(160KB)
PowerPC	―	―	185974(185KB)
Fat	―	―	277068(277KB)

表3(単位はバイト数)

68K	241627(241KB)
PowerPC	872264(872KB)
Fat	1105143(1.1MB)

リストと表

●1億回FOR～NEXT

```
DIM i&,t&
window OFF
WINDOW #1,"TEST",(50,50)-(320,240),_doc
t& = FN TICKCOUNT
FOR i&=1 to 100000000
NEXT
PRINT FN TICKCOUNT - t&
STOP
```

Runtime	FB1	FB2	FB3
68K	223	226	287
PowerPC	―	―	159

●1億回整数加算

```
DIM i&,t&,a&
window OFF
WINDOW #1,"TEST",(50,50)-(320,240),_doc
t& = FN TICKCOUNT
FOR i&=1 to 100000000
a& = a& + 1
NEXT
PRINT FN TICKCOUNT - t&
STOP
```

Runtime	FB1	FB2	FB3
68K	290	287	339
PowerPC	―	―	183

●100万回浮動小数点数加算

```
DIM i&,t&,a#
WINDOW OFF
WINDOW #1,"TEST",(50,50)-(320,240),_doc
t& = FN TICKCOUNT
FOR i&=1 TO 1000000
a# = a# + 0.0001
NEXT
PRINT FN TICKCOUNT - t&
STOP
```

Runtime	FB1	FB2	FB3
68K	403	408	97
PowerPC	―	―	7

●100万回浮動小数点数演算

```
DIM i&,t&,a#
WINDOW OFF
WINDOW #1,"TEST",(50,50)-(320,240),_doc
t& = FN TICKCOUNT
FOR i&=1 TO 1000000
a# = (a# * 1.03)/0.11 + 1.09 - 0.00025
a# = SIN(a#) * COS(1-a#)
NEXT
PRINT FN TICKCOUNT - t&
STOP
```

Runtime	FB1	FB2	FB3
68K	134559	135309	1825
PowerPC	―	―	87

●1億回PEEK

```
DIM i&,t&,a&,b&
WINDOW OFF
WINDOW #1,"TEST",(50,50)-(320,240),_doc
t& = FN TICKCOUNT
FOR i&=1 TO 100000000
b& = PEEK(@a&)
NEXT
PRINT FN TICKCOUNT - t&
STOP
```

Runtime	FB1	FB2	FB3
68K	367	366	391
PowerPC	―	―	184

●1億回PEEK

```
DIM i&,t&,a&,b&
WINDOW OFF
WINDOW #1,"TEST",(50,50)-(320,240),_doc
t& = FN TICKCOUNT
FOR i&=1 TO 100000000
POKE @a&,0
NEXT
PRINT FN TICKCOUNT - t&
STOP
```

Runtime	FB1	FB2	FB3
68K	341	340	418
PowerPC	―	―	287

＊計測環境
PowerMac 8500/132 + G3/233 Card
Memory 384MB

JavaScriptから始めるプログラミング　第4回
オブジェクトの作成方法とプログラムの再利用

古籏一浩　FuRuhata KaZuhiro

今回は独自のオブジェクトの作成方法とプログラムの再利用について説明します。簡単に再利用するには外部ファイルとしてJavaScriptプログラムを用意するだけ……で終わらないのが悲しいところです。再利用パッケージ作成のための注意点などをまとめてみましょう。

オブジェクトの作成

JavaScriptでは既製のオブジェクトがいくつか用意されており通常は，そのオブジェクトを利用してプログラミングを行います。たとえば画面に文字を表示する場合はdocumentオブジェクトのwriteメソッドを使って以下のように記述します。

document.write("Oh!mz");

これと同様に自分で作成したオブジェクトも，

オブジェクト名.メソッドまたはプロパティ

といったようにして利用することができます。自作したオブジェクト名がohxでありプロパティ名がvolであれば以下のようになります。

ohx.vol

このohxオブジェクトのvolプロパティに4を入れるのであれば以下のようになります。

ohx.vol = 4

既存のオブジェクトの使用方法とまったく同様です。

それでは実際にオブジェクトを自作してみましょう。リスト1が独自のオブジェクトを作成するスクリプトです。パターンとしては以下のようになります。

1)　雛形を用意する
2)　newを使ってオブジェクトを作成する

1)の雛形ですが，これはfunction命令を使います。そう，関数の作成とまったく同じです。ただし，中身の記述はやや異なります。中身には「プロパティ」「メソッド」を指定(作成)する命令が入ります。なんだか難しそうですが，thisを使ってプロパティ，メソッドに値を代入するだけという至ってシンプルな構造になっています。

プロパティもメソッドも，

this["プロパティ名またはメソッド名"] = 値
this.プロパティ名またはメソッド名 = 値

の2つのうちいずれかの方法で値を設定します。リスト1を見てみると以下の部分でプロパティに値を代入しているのがわかります。

```
this["name"]    = "FuRuhata"
this["age"]     = 30
this["address"] = "長野県塩尻市"
```

プロパティ名は，どうやってつけるのか？　というとthis～で宣言さえすれば自動的に作成され使用できるようになります。もし，宣言済みの名前があれば宣言済みのプロパティを参照します。

上記の記述は以下のようにしてもまったく同じです。

```
this.name      = "FuRuhata"
this.age       = 30
this.address   ="長野県塩尻市"
```

実際は連想配列になっているためthis.age = 30としたあとで，

this[1] = 29

とすればageの値が書き換わります。つまり上記3つのプロパティは以下のようにして参照することもできます(全然おすすめできませんが)。

```
this.name    は　this[0]
this.age     は　this[1]
this.address は　this[2]
```

通常はthis.ageなどわかりやすいように書いておくべきです。this [0] と

リスト1

```
<HTML>
<HEAD>
<TITLE>独自のオブジェクトを作成</TITLE>
<SCRIPT Language="JavaScript">
<!--
//  ここは雛形
function moto()
{
   this["name"]    = "FuRuhata"
   this["age"]     = 30
   this["address"] = "長野県塩尻市"
}

//  ここで実際のオブジェクトを作成
myObject = new moto();
// -->
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT Language="JavaScript">
<!--
document.write("myObject.name    = "+myObject.name+"<BR>");
document.write("myObject.age     = "+myObject.age+"<BR>");
document.write("myObject.address = "+myObject.address+"<BR>");
// -->
</SCRIPT>
</BODY>
</HTML>
```

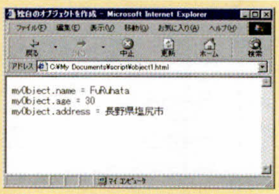


図1　リスト1の実行結果

リスト2

```
<HTML>
<HEAD>
<TITLE>独自のオブジェクトを作成</TITLE>
<SCRIPT Language="JavaScript">
<!--
//  ここは雛形
function moto()
{
   this.name    = "FuRuhata"
   this.age     = 30
   this.address = "長野県塩尻市"
}

//  ここで実際のオブジェクトを作成
myObject = new moto();
// -->
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT Language="JavaScript">
<!--
for(i in myObject)
{
   document.write("myObject."+i+" = "+myObject[i]+"<BR>");
}
// -->
</SCRIPT>
</BODY>
</HTML>
```

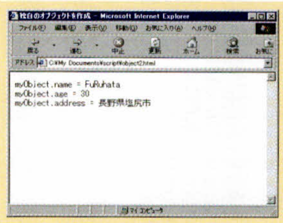


図2　リスト2の実行結果

書かれてもあとでわからなくなってしまいます。

先ほど連想配列になっていると書きましたが，JavaScriptにはオブジェクトの中身を表示する命令が用意されています。for (in) という命令です。リスト2が実際のサンプルです。この命令を使うと正式に公開されていない隠しメソッドやプロパティも見ることができます。

話がそれてしまったので，オブジェクトの作成に戻ります。雛形を用意しただけではオブジェクトは作成されません。実際のオブジェクトを作成するにはnew命令を使って宣言する必要があります。これは以下のようになります。

```
オブジェクト名 = new 雛形の名前
```

リスト1では，

```
myObject = new moto();
```

の部分が該当します。これによりオブジェクトを作成してしまえば雛形で宣言したプロパティを参照することができます。先ほど宣言したname，age，addressは以下のように記述されます。

```
myObject.name
myObject.age
myObject.address
```

雛形を基にして，いくつでも違う名前のオブジェクトを作成することができます。

```
a = new moto()
b = new moto()
c = new moto()
```

nameを参照するには以下のようになります。

```
a.name
b.name
c.name
```

異なる値を持つオブジェクトの作成

リスト1の雛形ではname，age，addressの値は，あらかじめ決定されていました。そのため作成したオブジェクトのプロパティを変更するには以下のように記述することになります。

```
myObject     = new moto()
myObject.age = 28
```

作成するオブジェクトが1個であれば毎回代入するのも苦痛ではありませんが，3つも4つも作成するとなると，そのたびに記述しなければならず大変です。つまり以下のようになってしまいます。

```
a = new moto()
a.age = 16
b = new moto()
b.age = 29
c = new moto()
c.age = 22
```

このように，あるプロパティの値が作成するオブジェクトにより異なる，不定である場合には雛形にパラメータとして値を渡すことができます。実際のプログラムはリスト3のようになります。

関数に必要なパラメータを与えて呼び出すのとまったく変わりません。リスト1と異なるのは，

```
this["age"] = n
```

だけです。これで渡されたパラメータをageプロパティに代入することができます。このようにすると以下のように簡素に記述することができるようになります。

```
myObject = new moto(28)
```

これで自動的にmyObject.ageには28が代入されます。パラメータはひとつでなく複数渡すことができます。また，パラメータの数が不定になることもあります。この場合は，

```
this.arguments.length
```

とすれば渡されたパラメータの数を知ることができます。

プロパティの追加

すべてのオブジェクトが同じプロパティとメソッドを持っているのであればリスト1，3のように雛形を作成しておくことで使い回しができます。しかし，実際にはそのようなことは少なく「似ているけど違う」場合がほとんどです。このような場合は，同じ部分だけを雛形として用意しておき，異なる部分だけあとで追加することができます。

実際のプログラムがリスト4です。myObject = new moto (30);のあとで以下のようにしてプロパティを追加します。

```
myObject["useMac"] = "PowerMac 8500/132";
```

追加するといっても難しいことはなにもなく，以下のように記述すれば自動的に追加され使用できるようになります。

```
オブジェクト名["プロパティ名"] = 値
オブジェクト名.プロパティ名 = 値
```

逆に追加したものを削除することはできません。もっとも，現在ではメモリも大量に使用できますから，あんまりケチケチせずに使ってもよいのでは

リスト3

```
<HTML>
<HEAD>
<TITLE>独自のオブジェクトを作成</TITLE>
<SCRIPT Language="JavaScript">
<!--
//  ここは雛形
function moto(n)
{
  this["name"]    = "FuRuhata"
  this["age"]     = n
  this["address"] = "長野県塩尻市"
}

//  ここで実際のオブジェクトを作成
myObject = new moto(30);
// -->
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT Language="JavaScript">
<!--
document.write("myObject.name    = "+myObject.name+"<BR>");
document.write("myObject.age     = "+myObject.age+"<BR>");
document.write("myObject.address = "+myObject.address+"<BR>");
// -->
</SCRIPT>
</BODY>
</HTML>
```



図3 リスト3の実行結果

リスト4

```
<HTML>
<HEAD>
<TITLE>独自のオブジェクトを作成</TITLE>
<SCRIPT Language="JavaScript">
<!--
//  ここは雛形
function moto(n)
{
  this["name"]    = "FuRuhata"
  this["age"]     = n
  this["address"] = "長野県塩尻市"
}

//  ここで実際のオブジェクトを作成
myObject = new moto(30);
//  プロパティを付け足す
myObject["useMac"] = "PowerMac 8500/132";
// -->
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT Language="JavaScript">
<!--
document.write("myObject.name    = "+myObject.name+"<BR>");
document.write("myObject.age     = "+myObject.age+"<BR>");
document.write("myObject.address = "+myObject.address+"<BR>");
document.write("myObject.useMac  = "+myObject.useMac+"<BR>");
// -->
</SCRIPT>
</BODY>
</HTML>
```

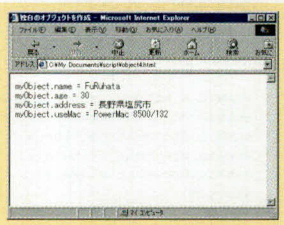


図4 リスト4の実行結果

ないかと思います。

オブジェクトを作成したあとにプロパティの値を変更するには以下のようになります。

オブジェクト名["プロパティ名"] = 値
オブジェクト名.プロパティ名 = 値

具体的にはmyObject.age = 22となります。実際のプログラムをリスト5に示します。

リスト5

```
<HTML>
<HEAD>
<TITLE>独自のオブジェクトを作成</TITLE>
<SCRIPT Language="JavaScript">
<!--
// ここは雛形
function moto(n)
{
  this["name"]    = "FuRuhata"
  this["age"]     = n
  this["address"] = "長野県塩尻市"
}

// ここで実際のオブジェクトを作成
myObject = new moto(30);
// プロパティを変更してみる
myObject["name"] = "古籏一浩";
// -->
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT Language="JavaScript">
<!--
document.write("myObject.name    = "+myObject.name+"<BR>");
document.write("myObject.age     = "+myObject.age+"<BR>");
document.write("myObject.address = "+myObject.address+"<BR>");
// -->
</SCRIPT>
</BODY>
</HTML>
```



図5　リスト5の実行結果

リスト6

```
<HTML>
<HEAD>
<TITLE>独自のオブジェクトを作成</TITLE>
<SCRIPT Language="JavaScript">
<!--
// ここは雛形
function moto()
{
  this["name"]    = "FuRuhata"
  this["age"]     = 30
  this["address"] = "長野県塩尻市"
  this["msg"]     = newMsg;
}
// メソッドの雛形
function newMsg(n)
{
  subWin = window.open("","Message","width=320,Cheight=240");
  subWin.document.open("text/html");
  subWin.document.bgColor = "white";
  subWin.document.write(n);
  subWin.document.close();
  alert(n)
}

// ここで実際のオブジェクトを作成
myObject = new moto();
// -->
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT Language="JavaScript">
<!--
document.write("myObject.name    = "+myObject.name+"<BR>");
document.write("myObject.age     = "+myObject.age+"<BR>");
document.write("myObject.address = "+myObject.address+"<BR>");
document.write("myObject.msg     = "+myObject.msg+"<BR>");

// メソッドを呼び出してみる
myObject.msg("文字よ出ろっ!!");
// -->
</SCRIPT>
</BODY>
</HTML>
```

メソッドの作成

オブジェクトのプロパティの作成方法については，だいたいわかったのではないかと思います。しかし，オブジェクトにはプロパティだけでなく指示を与えて実行するメソッドがあります。プロパティしか使えないオブジェクトでは魅力が半減してしまいます。そこで，今度はメソッドの作成方法について説明します。

実際のプログラムはリスト6のようになります。まず，雛形の部分はほとんど変わりません。唯一追加されたのが以下の行です。

this["msg"] = newMsg;

これが呼び出すメソッド名と呼び出す関数名の指定部分です。msgがメソッド名，newMsgが呼び出される関数名になります。newMsg関数は別に作成しておきます。上記指定で間違って以下のように指定してはいけません。

this["msg"] = newMsg ();

これだとnewMsgが実行されて，その戻り値がmsgに代入されてしまいます。呼び出す前に実行されたのでは困ります。

作成したメソッドは以下のようにして呼び出します。まず，

myObject = new moto ();

としてオブジェクトを作成。次にメソッドを呼び出します。

myObject.msg ("文字よ出ろっ!!");

これによりnewMsg関数が呼び出されて実行されます。パラメータも渡すことができますが，雛形の部分で渡すのではなく実行される関数側で受けることになります。

外部ソースファイル

Netscape 3.0，Internet Explorer 4.0以降ではJavaScriptプログラムを切り離して外部のソースファイルとして使用することができます。外部ソースファイルには，純粋にJavaScriptプログラムのみ記述します。<SCRIPT>タグなどHTMLタグは一切不要です。注意点はファイル名の拡張子を.jsにしておかないと読み込んでくれないことです。

リスト6の雛形部分を切り離したものがリスト7です。外部JavaScriptファイルを読み込むには以下のように記述します。

<SCRIPT Language = "JavaScript"

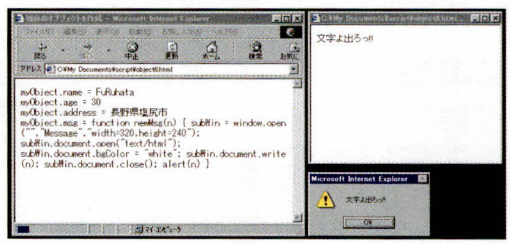


図6
リスト6の実行結果

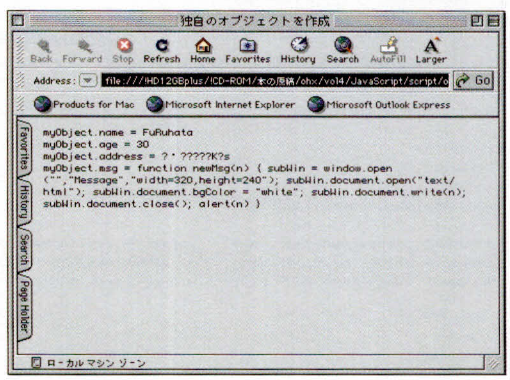


図7
Mac版 Internet Explorer 4.xでは日本語（address部分）が文字化けしてしまう

SRC = "ファイル名"></SCRIPT>

このように外部ソースファイルにしておくとプログラムのミスも防ぐことができ，デバッグにかかる時間も軽減されます。

しかし，こんなに便利な外部ソースファイルですがInternet Explorerの都合により二重にファイルを準備しなければならない事態に陥ってしまっています。Mac版のInternet Explorer 4.xでは外部ソースファイルに日本語が含まれていると文字化けしてしまい，正しく表示されないことがあります*1。正しく表示するにはテキストコードをUnicode（UTF-8）にしなければなりません。これ以外では文字化けしてしまい使いものにならないことがあります（図7）。

このような状態なのでページ内で外部ソースファイルを使用する場合は，以下のような手法を取る必要があります。

1）外部ソースファイル使用前のページでブラウザと機種により分岐させる
2）日本語を外部ソースファイルに含まない
3）ページ内でブラウザと機種ごとに読み込ませる

1）の手法を取っているサイトは多数ありますのでいちばん現実的といえます。しかし，作成する手間を考えると面倒です。

リスト7（本体）

```
<HTML>
<HEAD>
<TITLE>独自のオブジェクトを作成</TITLE>
<SCRIPT Language="JavaScript" SRC="object.js"></SCRIPT>
<SCRIPT Language="JavaScript">
<!--
//  ここで実際のオブジェクトを作成
myObject = new moto();
// -->
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT Language="JavaScript">
<!--
document.write("myObject.name      = "+myObject.name+"<BR>");
document.write("myObject.age       = "+myObject.age+"<BR>");
document.write("myObject.address = "+myObject.address+"<BR>");
document.write("myObject.msg       = "+myObject.msg+"<BR>");

//  メソッドを呼び出してみる
myObject.msg("文字よ出ろっ!!");
// -->
</SCRIPT>
</BODY>
</HTML>
```

リスト7（外部ソースファイル）

```
//  ここは雛形
function moto()
{
  this["name"]    = "FuRuhata"
  this["age"]     = 30
  this["address"] = "長野県塩尻市"
  this["msg"]     = newMsg;
}
//  メソッドの雛形
function newMsg(n)
{
  subWin = window.open("","Message","width=320,height=240");
  subWin.document.open("text/html");
  subWin.document.bgColor = "white";
  subWin.document.write(n);
  subWin.document.close();
  alert(n)
}
```

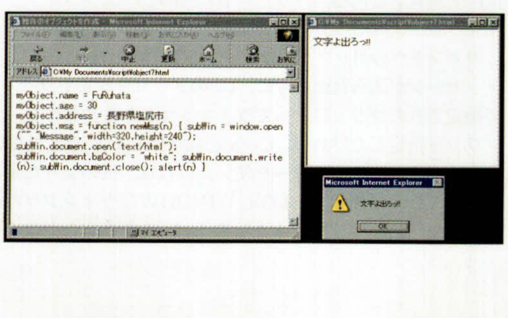


図8
リスト7の実行結果

2）の手法は確実かつ現実的といえます。日本語を含む部分はHTML文書本体に記述すれば問題ありません。

3）の手法は数行の記述で可能で外部ソースファイルに日本語を含む場合でも問題なく動作します。実際のプログラムはリスト8になります。別々に外部ソースファイルを読み込ませる部分は以下のようになります。

```
wrtSrc = "object.js";
if ((document.all) &&
(navigator.userAgent.toUpperCase().indexOf("MAC") >
                                0)  wrtSrc = "unicode.js";
document.write("<SCRIPT
                SRC="+wrtSrc+"></"+"SCRIPT>");
```

MacintoshでInternet Explorer 4.0/4.5の場合のみUnicode版の外部ソースファイルを指定しています。document.writeで<SCRIPT>タグを書き出して強引に読み込ませています。欠点としては<BODY>～</BODY>タグ内に記述することになるため，ページが完全に読み込まれていない状態でユーザーにインタラクティブな操作が行われるとJavaScriptエラーを引き起こす場合もあります。

どの手法を使うかは自由ですが，安全を追うならば2)，そうでなければ3）の手法を取るのがよいのではないかと思います。

＊1　Windows版のInternet Explorer 5.01/5.5でも同様の症状に陥ることがあります。

結局どう使うの？

独自オブジェクトの作成方法を書きましたが「結局どこで，どう使うの？」という人もいると思います。当然使わなくてもプログラムを作成できますしなにも問題ありません。では，いったいなにに使うのか？　それは「ゲーム」です。ゲームの場合は，かなり多くのパラメータを必要とするためいちいち配列宣言してデータを読み書きするというのは，ある段階を超えるとプログラムが大変（非現実的）になります。ロールプレイングゲーム（RPG）などは特に顕著です。ゲームには複数の敵が出現し動きや体力，パワーなどさまざまです。その中で表示座標や体力，パワーなどは共通して使用することができます。これに関しては，また次回。

ということで，次回はJavaScriptでレイヤーを使ってゲームを作成します。

リスト8（本体）

```
<HTML>
<HEAD>
<TITLE>独自のオブジェクトを作成</TITLE>
</HEAD>
<BODY>
<SCRIPT Language="JavaScript">
<!--
// Mac+IE4.5だけ別の外部ソースファイルを読み込む
wrtSrc = "object.js";
if  ((document.all)  &&  (navigator.userAgent.toUpperCase().indexOf
("MAC") > 0)) wrtSrc = "unicode.js";
document.write("<SCRIPT SRC="+wrtSrc+"></"+"SCRIPT>");
//-->
</SCRIPT>
<SCRIPT Language="JavaScript">
<!--
//  ここで実際のオブジェクトを作成
myObject = new moto();
// -->
</SCRIPT>

<SCRIPT Language="JavaScript">
<!--
document.write("myObject.name      = "+myObject.name+"<BR>");
document.write("myObject.age       = "+myObject.age+"<BR>");
document.write("myObject.address = "+myObject.address+"<BR>");
document.write("myObject.msg       = "+myObject.msg+"<BR>");

//  メソッドを呼び出してみる
myObject.msg("文字よ出ろっ!!");
// -->
</SCRIPT>
</BODY>
</HTML>
```

VisualC++で始めるWindowsプログラミング（4）
MP3プレイヤーを作ろう

菊地 功　Kikuchi Isawo

すでに音楽再生環境として定着して久しいMP3形式の音楽データ。プレイヤーもたくさん発表されているが，どうも自分の要求にあわないということもあるだろう。気に入った仕様のものがなければ自分で作ってしまえばいい。Windows Media SDKを使えばなにも考えなくてもMP3データの再生が可能だ。ウィンドウのスキニングなど，流行りのフィーチャも取り入れてみる。

うちにはMP3専用マシンがある（正確には，兼DVDプレイヤー）。話せば長くて短いのだが，3連装のCDステレオがあって，そのCDの入れ換えが結構面倒なのだ。この3連装ってのが曲者で，いっそのこと単装ならばセコセコ入れ替えるし，5連装ならば旬なCDはほぼすべて入るのだが，いかんせん中途半端で，結局同じ3枚のCDを聞き続けることになる。だいたい，いちいちCDをケースから出し入れするのが面倒臭い。ってなわけで，BGMマシンを組んだわけだ。手持ちのCDを吸い出して当時流行りのMP3に変換し，MP3プレイヤーで再生する。CDステレオのほうは，そのアンプとして利用されることになる。ソフトならば，CDの枚数制限なんてないに等しいし，CDを出し入れする手間もないはずである。

が，……甘かった。というのは，一般的なMP3プレイヤーは1曲単位で操作されるものだからだ。CDステレオの代替品として利用したい私としては，CD単位でプレイヤーに出し入れしたい。プレイリストに対応したプレイヤーによっては，そのプレイリストをドロップすればすべての曲を追加できるものもあるが，結局プレイヤー内部でプレイリスト内の曲が展開されてしまい，削除は1曲1曲行うしかない。私は終始CD単位での操作を行いたいのだぁ〜っ！　ついでにいえば，CDのタイトルだけではピンとこないので，そのジャケットを表示してほしい。これは，CDを選ぶときもそうだし，現在再生しているCDを識別するためにも必要である。そんなわけで，マシンを組んでからもうずいぶん経つが，なにか作らにゃいかんかってことになったんである。

とまあ，以上のような漠然とした仕様要求があるわけだが，それよりも前に，MP3プレイヤーといえばやはりスキンである。なんでかわからないけど，そうなんである。スキンっていっても，ちゃんとした定義があるわけではない（と思う）。ただ単にウィンドウの背景に画像を貼り付けられるだけでスキンだといい張っているものから，ウィンドウ全体を覆い，スキン画像によってウィンドウの形が変わり，しかもユーザーが変更できるといったものまで，何段階かあると思われる。ま，一般的にいって，次の3つの要素ではないだろうか。

1）タイトルバーなどを含むウィンドウイメージを画像で供給
2）非矩形ウィンドウ
3）スキン画像の変更が可能

このうち，3）についてはユーザーインタフェイスの問題なので置いとくとして，まずは1）と2）について実験してみよう。この辺りはMP3プレイヤーとは関係なしに応用できる部分でもある。

ところで，前回まではこの連載ではMFCを使ってきた。MFCは便利である半面，ちょっと深い部分をいじろうとすると，とたんに面倒になる。今回のスキンでは，MFCを使うと二度手間になるうえ，無駄にサイズが大きくなり，その割にはたいした恩恵を受けられないということで，MFCは使わないことにする。そんなわけで，まずは一般的なウィンドウを作るまでの基本解説をすることにしよう。

ウィンドウができるまで

MFCを使っていれば，プロジェクトを作った段階で生成されるスケルトンをビルドするだけで，とりあえずウィンドウが開くだけのアプリケーションが作られる。しかし，そうでない場合には，ひとつのウィンドウを作るまでにもいくつかのステップが必要である。

1）ウィンドウクラスの登録

ウィンドウクラスはいまから作るウィンドウの基本仕様書みたいなものだ。これをまずシステムに提出する必要がある。次のような感じだ。

```
WNDCLASSEX wc;
wc.cbSize = sizeof(WNDCLASSEX);
wc.style = 0;
wc.lpfnWndProc = (WNDPROC)MainWndProc;
wc.cbClsExtra = 0;
wc.cbWndExtra = 0;
wc.hInstance = hInstance;
wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
wc.hCursor = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
wc.lpszMenuName = NULL;
wc.lpszClassName = "FooWindowClass";
wc.hIconSm = NULL;
RegisterClassEx(&wc);
```

WINDCLASSEX構造体に必要事項を記入のうえ，RegisterClassEx()窓口に提出する。WINDCLASSEX構造体のメンバの詳しい説明は各自でヘルプで調べてもらうとして，重要な部分だけをピックアップしよう。

lpfnWndProc

ウィンドウにメッセージが送られたときに，それを処理するウィンドウプロシージャを指定する。詳しくは後述する。

hIcon

（システムメニューを持つ）ウィンドウの左上に表示されるアイコンを指定する。上の例では特にアイコンを作らずに，システムのアプリケーションアイコンを使っている。もしアプリケーションが自前のアイコンをリソースに持っているのなら，

```
LoadIcon(hInstance, MAKEINTRESOURCE(IDI_ICON));
```

などとすればよい（IDI_ICONはアイコンのID）。

hCursor

マウスカーソルがこのウィンドウの上にきたときに表示するカーソルを指定する。特にカーソルを変化させる必要がない場合は，上のようにシステムの矢印カーソルを指定しておけばよい。自前のカーソルを表示したい場合は，アイコンと同様にリソースIDを指定する。

hbrBackground

ウィンドウのバックグラウンドブラシを指定する。WM_ERASEBKGNDメッセージが送られたときに，（このメッセージをトラップしなければ）ここで指定されたブラシによってウィンドウがクリアされる。ユーザーが任意のブラシを指定したければ，CreateSolidBrush()などで作成したブラシを指定するが，システムカラーを使いたければ，例のような定数に1を加えた値を使うことができる（COLOR_WINDOWはウィンドウの色，標準では白）。なぜ1を足さなければならないのかは，0を指定した場合はWM_ERASE

BKGNDは処理されない（自分で処理する必要がある）という仕様によるためだろう。ちなみにWINUSER.Hを覗くと，

```
#define COLOR_SCROLLBAR       0
#define COLOR_BACKGROUND      1
#define COLOR_ACTIVECAPTION   2
#define COLOR_INACTIVECAPTION 3
    :
```

のように宣言されている。きっと「文字定数は0から設定すべし」とかなんとかいうポリシーでもあったのだろう。

lpszMenuName

メニューが必要ならば，リソースのメニューIDをMAKEINTRESOURCE () した値を指定する。

lpszClassName

ウィンドウのクラス名を指定する。ほかで使われていないユニークなものでないと，RegisterClassEx () 関数は失敗する（かもしれない）。

2）ウィンドウのクリエイト

登録したウィンドウクラスに基づいて，ウィンドウのクリエイトを行う。

```
CREATESTRUCT cs;
cs.dwExStyle = 0;
cs.lpszClass = "FooWindowClass";
cs.lpszName = "Foo Application";
cs.style = WS_OVERLAPPEDWINDOW|WS_VISIBLE;
cs.x = CW_USEDEFAULT;
cs.y = CW_USEDEFAULT;
cs.cx = CW_USEDEFAULT;
cs.cy = CW_USEDEFAULT;
cs.hwndParent = NULL;
cs.hMenu = NULL;
cs.hInstance = hInstance;
cs.lpCreateParams = NULL;
HWND hWnd = CreateWindowEx (cs.dwExStyle,
cs.lpszClass, cs.lpszName, cs.style, cs.x, cs.y, cs.cx,
cs.cy, cs.hwndParent, cs.hMenu, cs.hInstance,
cs.lpCreateParams);
```

CREATESTRUCT構造体は特に必要ないのだが，CreateWindowEx () 関数に渡す引数が，CREATESTRUCTのメンバとまったく同じなので，説明上まずCREATESTRUCT構造体を作ってみた。

dwExStyle

後述のstyleでは指定されない，ちょっと特殊な拡張スタイルの指定ができるが，内容については各自で調べてほしい。なお，このスタイルを指定しないのであれば，CreateWindowEx () ではなくCreateWindow () 関数を使用してもよい。

lpszClass

先ほど登録したウィンドウクラス名を指定する。

lpszName

キャプションバーに表示される文字列を指定する。

style

ウィンドウスタイルを指定する。WS_OVERLAPPEDWINDOWとは，一般的なトップレベルウィンドウであり，WS_VISIBLEは可視状態であることを示す。WS_VISIBLEを指定しない場合は，ShowWindow () 関数で可視状態にするまで不可視となる。

x,y,cx,cy

ウィンドウの表示位置（左上）とサイズである。CW_USEDEFAULTを指定すると，システムが適当に決める。

hwndParent

親のウィンドウハンドル。トップレベルウィンドウの場合は，NULLにする。

hMenu

メニューハンドルを指定する。ウィンドウクラスで指定したメニューではなく，別のメニューを表示したい場合に使用する。また，チャイルドウィンドウの場合は，コントロールIDを指定できる。

lpCreateParams

CreateWindowEx () 関数をコールすると，まずWM_NCCREATEとWM_CREATEメッセージがウィンドウプロシージャに送られる。その際に渡したいポインタをユーザーが任意に設定できる。具体例は後ほど。

3）メッセージポンプとトラップ

自分（アプリケーション）宛のメッセージがあるかどうかを調べ，もしあれば適切なウィンドウプロシージャにメッセージを転送するのがメッセージポンプである。とはいえ，これは，

```
MSG msg;
while (GetMessage (&msg, NULL, 0, 0)) {
    TranslateMessage (&msg);
    DispatchMessage (&msg);
}
```

といった記述が常套手段である。ダイアログなどがあると，IsDialogMessage () をかませる必要があったりするが，基本的にはこれだけだ。アプリケーションが動いている間はこのループは回り続けるので，このループのあとにはアプリケーションの終了処理を記述して，メインルーチンは終了となる。ちなみにWindowsアプリケーションのメインルーチンはWinMain () である。

さて，転送されたメッセージの処理だが，これはウィンドウクラスの登録時に指定したMainWndProc () の仕事である（関数名はなんでもいい）。まず処理が必要なのは，WM_CREATEとWM_DESTROYだろう。それにウィンドウになにか描画したいのなら，WM_PAINTもトラップする。switch分で振り分けるのが一般的だ。

```
LONG APIENTRY MainWndProc (HWND hWnd,
                UINT message, UINT wParam, LONG lParam)
{
    switch (message) {
    case WM_CREATE:
        OnCreate (hWnd);
        return 0;
    case WM_DESTROY:
        OnDestroy ();
        PostQuitMessage (0);
        return 0;
    case WM_PAINT:
        OnPaint (hWnd, (HDC) wParam);
        return 0;
    }
    return DefWindowProc (hWnd, message, wParam,
    lParam );
}
```

引数として渡されるwParamとlParamは，メッセージ (message) によって内容が異なる。たとえば，WM_PAINTメッセージでは，wParamには描画されるべきデバイスコンテキストハンドルが渡される。念のためにいっておくと，WM_CREATEはウィンドウがクリエイトされた直後に送られ（ウィンドウがクリエイトされるときに初めて送られるメッセージはWM_NCCREATE），WM_DESTROYはウィンドウが破棄された直後に送られるメッセージである。ここではとりあえずMFCに習って各関数に飛ばすだけで，その処理内容については省略しよう。基本的にWM_CREATEではウィンドウの初期化処理，WM_DESTROYでは終了処理をすればいい。WM_DESTROYではPostQuitMessage (0) を呼んでいるが，これはアプリケーションに終了メッセージを送ることを意味する。したがって，トップレベルウィンドウ以外はこの関数は省くべきだろう。それぞれのメッセージを処理したあとの戻り値は，たいていは0でいいのだが，メッセージによってはなんらかの値を返す必要のあるものもある。また，メッセージを処理しない場合は，DefWindowProc () に渡すことで，デフォルトの

処理をさせることができる。

とまあ，この3つのステップでウィンドウが作成されることになる。例がリスト1（RgnWindow.cpp：サンプルRgnWindow）である。ここでは（C++言語のほうの）ウィンドウのクラスRgnWndを定義し，InitInstance（）でそのインスタンスを作成，コンストラクタ内でウィンドウクラスの登録を行っている。さらにRgnWndのメンバCreate（）関数を呼び，そこでウィンドウをクリエイトする。

その際，CreateWindowEx（）関数の最後の引数，ユーザー定義のデータとして，thisポインタ，すなわちRgnWndのインスタンスポインタを渡している。こうすることで，MainWndProc（）にWM_NCCREATE（またはWM_CREATE）が送られたとき，その引数としてlParamでLPCREATESTRUCT構造体へのポインタが渡され，そのlpCreateParamsメンバにRgnWndのインスタンスが入っていることになる。

ところで，このMainWndProc（）は先ほどとはちょっと違う。なにやらWM_NCCREATEでそのポインタを取り出し，SetWindowLong（）という関数に入れている。ウィンドウには1つひとつそれぞれの属性を管理する領域がある。たとえばスタイルやプロシージャ，IDなどだ。その内部に，ユーザーが自由に利用できる32ビット領域も含まれている。これがGWL_USERDATAで示される領域である。ここに対応しているRgnWndのインスタンスポインタを入れて，管理しようというわけだ。以降，メッセージはそのポインタからRgnWnd::WndProc（）に送るようにし，そちらで処理している。グローバルで，

リスト1

```
// RgnWindow.cpp : アプリケーション用のエントリ ポイントの定義
//

#include "stdafx.h"
#include "RgnWnd.h"

#define APPNAME "RgnWindow"
static char szAppName[] = APPNAME;
static char szTitle[]   = APPNAME;

RgnWnd *prgnWnd;

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow )
{
  // TODO: この位置にコードを記述してください。
  if( !InitInstance( hInstance ) ){
      MessageBox( NULL, "アプリケーションの初期化に失敗しました。", "RgnWindow",
MB_OK|MB_ICONHAND );
      return FALSE;
  }

  MSG msg;
  while( GetMessage( &msg, NULL, 0, 0 ) ){
      TranslateMessage( &msg );  // Translates virtual key codes.
      DispatchMessage( &msg );   // Dispatches message to window.
  }
  ExitInstance();
  return msg.wParam; // Returns the value from PostQuitMessage.
}

BOOL InitInstance( HINSTANCE hInstance )
{
  prgnWnd = new RgnWnd( hInstance );
  HWND hWnd = prgnWnd->Create();
  return hWnd?TRUE:FALSE;
}

void ExitInstance()
{
  delete prgnWnd;
}

LONG APIENTRY MainWndProc( HWND hWnd, UINT message, UINT wParam, LONG lParam )
{
  if( message==WM_NCCREATE ){
      CREATESTRUCT *pcs = (CREATESTRUCT*)lParam;
      SetWindowLong( hWnd, GWL_USERDATA, (LONG)pcs->lpCreateParams );
  }
  RgnWnd *pWnd = (RgnWnd*)GetWindowLong( hWnd, GWL_USERDATA );
  if( pWnd==NULL ){
      return DefWindowProc( hWnd, message, wParam, lParam );
  } else {
      return pWnd->WndProc( hWnd, message, wParam, lParam );
  }
  return 0;
}

//////////////////////////////////////////////////////////////////////
// RgnWnd クラス
//////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////
// 構築/消滅
//////////////////////////////////////////////////////////////////////

RgnWnd::RgnWnd( HINSTANCE hInstance )
{
   WNDCLASSEX wc;
   wc.cbSize = sizeof( WNDCLASSEX );
   wc.style = 0;
   wc.lpfnWndProc = (WNDPROC)MainWndProc;
   wc.cbClsExtra = 0;
   wc.cbWndExtra = 0;
   wc.hInstance = hInstance;
   wc.hIcon = LoadIcon( NULL, IDI_APPLICATION );
   wc.hCursor = LoadCursor( NULL, IDC_ARROW );
   wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
   wc.lpszMenuName = NULL;
   wc.lpszClassName = szAppName;
   wc.hIconSm = NULL;
   RegisterClassEx( &wc );
   m_hInstance = hInstance;
}

RgnWnd::~RgnWnd()
{
}

HWND RgnWnd::Create()
{
   CREATESTRUCT cs;
   cs.lpCreateParams = this;
   cs.hInstance = m_hInstance;
   cs.hMenu = NULL;
   cs.hwndParent = NULL;
   cs.cy = CW_USEDEFAULT;
   cs.cx = CW_USEDEFAULT;
   cs.y = CW_USEDEFAULT;
   cs.x = CW_USEDEFAULT;
   cs.style = WS_OVERLAPPEDWINDOW|WS_VISIBLE;
   cs.lpszName = szTitle;
   cs.lpszClass = szAppName;
   cs.dwExStyle = 0;
   m_hWnd = CreateWindowEx( cs.dwExStyle, cs.lpszClass, cs.lpszName, cs.style,
       cs.x,  cs.y,  cs.cx,  cs.cy,  cs.hwndParent,  cs.hMenu,  cs.hInstance,
cs.lpCreateParams );
   return m_hWnd;
}

LRESULT RgnWnd::WndProc( HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam )
{
   switch( message ){
   case WM_CREATE:
       OnCreate( hWnd );
       return 0;
   case WM_DESTROY:
       OnDestroy();
       PostQuitMessage( 0 );
       return 0;
   case WM_SIZE:
       OnSize( hWnd, wParam, LOWORD(lParam), HIWORD(lParam) );
       return 0;
   }
   return DefWindowProc( hWnd, message, wParam, lParam );
}

void RgnWnd::OnCreate( HWND hWnd )
{
   OnSize( hWnd, SIZE_RESTORED, 0, 0 );
}

void RgnWnd::OnDestroy()
{
}

void RgnWnd::OnSize( HWND hWnd, int fwSizeType, int nWidth, int nHeight )
{
   if( fwSizeType==SIZE_MAXIMIZED || fwSizeType==SIZE_RESTORED ){
       RECT rect;
       GetWindowRect( hWnd, &rect );
       HRGN hRgn = CreateEllipticRgn( 0, 0, rect.right-rect.left, rect.bottom-
rect.top );
       SetWindowRgn( hWnd, hRgn, TRUE );
   }
}
```

```
RgnWnd *prgnWnd;
```

として宣言してしまっているのだから，そちらを使ってもいいのだが，このほうがより汎用的であり，ひとつのウィンドウクラスで複数のウィンドウを作成した場合も適切な処理が行える。また，ウィンドウクラスを登録するときに，メッセージを直接RgnWnd::WndProc ()に送るようにしてしまえばいいと思うかもしれないが，それはあまりうまくない。以前クラスのメソッドというのは「第0引数として暗黙にthisポインタが渡される」と述べた。そして，メンバへのアクセスは，このthisポインタが利用されている。もし直接RgnWnd::WndProc ()へメッセージを渡したら，このthisポインタが渡されず，結局そのままではクラスメンバへ正しくアクセスされないことになる。少々面倒だが，この方法がいちばんスマートではないだろうか。

さて，そのRgnWnd::WndProc ()だが，とりあえずなにも表示しない代わりに，非矩形ウィンドウにしてみよう。方法は実に簡単である。領域を示すリージョンのハンドルを作り，それをSetWindowRgn ()関数に渡すだけだ。リージョンを作る関数はいくつかあるが，ここでは楕円領域を作るCreateEllipticRgn ()を使った。引数は，楕円の左上と右下の座標だ。SetWindowRgn ()で設定される領域はウィンドウ座標なので，左上は0,0，右下はそれぞれウィンドウの幅と高さとなる。ウィンドウのサイズを変更すると，リージョンもそれにあわせて変更させたいので，WM_SIZEメッセージをトラップし，ウィンドウサイズが変更されるたびにリージョンを作り直し，ウィンドウにセットしている。

これを実行したのが，図1だ（サンプルRgnWindow）。普通の四角いウィンドウを丸く切り抜いただけなので，キャプションバーやウィンドウフレームがトホホなことになっているが，切り取られた四隅の部分からは，ちゃんとマウスで下のウィンドウやデスクトップを操作することができる。また，上下左右に少しだけ残っているフレームをマウスでつかんでリサイズすれば，それに応じて円の大きさも変わる。なお，システムメニューもクローズボックスも隠れてしまったが，このウィンドウをアクティブにしてAlt+F4キーを押せば終了できる（あるいはキャプションバーで右クリックするとシステムメニューが表示される）。

スキンと非矩形ウィンドウ

ずいぶん下準備が長くなってしまったが，ようやく本番である（スキンの）。基本仕様としては，用意されたスキン画像をウィンドウに貼り付けるとともに，その画像の輪郭をリージョンとして抽出し，ウィンドウにセットする。そんなわけで用意した画像が図2である。8ビットカラーで，背景部分はパレット0で描かれている。これをビットマップリソースに加える。

さてリージョンだが，今度はCreateRectRgn ()を使う。これは単純な長方形のリージョンを作成する関数だ。これひとつではあまりうまみはないが，リージョンというのは複数のリージョンを合成することができるのだ。つまり，1ラスタごとに矩形リージョンを作り，それを結合すれば，理論的にはどんな形のリージョンでも実現できることになる（リージョンを結合するとどんどん重くなっていくため，実用上は結合数に限界があるだろう）。というわけで，リソースのビットマップからリージョンを作る部分がリスト2である（RgnWindow.cpp抜粋：サンプルRgnWindow2）。WM_CREATEメッセージを処理する初期化部分で行っている。ビットマップハンドルからビットマップビットを取得するまではナンだが（素直にLoadResource ()しろという話もあるが），まあ要するに8ビットで取得したら0を除く部分を探し，それをUpdateRgnメソッドで作成・結合しているわけだ（ちょっとはしょりすぎか？）。ちなみにCombineRgn ()の引数は，

```
int CombineRgn(
    HRGN hrgnDest,
    HRGN hrgnSrc1,
    HRGN hrgnSrc2,
    int fnCombineMode
);
```

となっていて，hrgnSrc1とhrgnSrc2をfnCombineModeで結合したものがhrgnDestに入れられる。RGN_ORというのは，いうまでもなくOR結

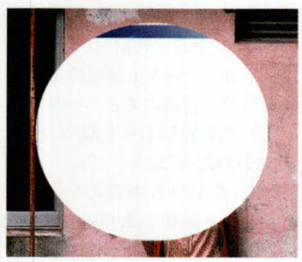


図1



図2

リスト2

```
void RgnWnd::OnCreate( HWND hWnd )
{
   m_hBitmap = LoadBitmap( m_hInstance, MAKEINTRESOURCE(IDB_OHX) );
   HDC hDC = GetDC( hWnd );
   BITMAPINFOHEADER *pbmih = (BITMAPINFOHEADER*)GlobalAlloc( GPTR,sizeof(BITMAP
INFOHEADER)+sizeof(RGBQUAD)*256 );
   memset( pbmih, 0, sizeof(BITMAPINFOHEADER) );
   pbmih->biSize = sizeof(BITMAPINFOHEADER);
   GetDIBits( hDC, m_hBitmap, 0, 0, NULL, (BITMAPINFO*)pbmih, DIB_RGB_COLORS );

   int x, y;
   m_nWidth = pbmih->biWidth;
   m_nHeight = pbmih->biHeight;
   int line = (m_nWidth+3)/4*4;
   LPBYTE lpvBits = (LPBYTE)GlobalAlloc( GPTR, line*m_nHeight );
   pbmih->biWidth = m_nWidth;
   pbmih->biHeight = m_nHeight;
   pbmih->biPlanes = 1;
   pbmih->biBitCount = 8;
   pbmih->biCompression = BI_RGB;
   pbmih->biXPelsPerMeter = pbmih->biYPelsPerMeter = 0;
   pbmih->biClrUsed = pbmih->biClrImportant = 256;
   pbmih->biSizeImage = 0;
   GetDIBits( hDC, m_hBitmap, 0, pbmih->biHeight, lpvBits, (BITMAPINFO*)pbmih,DIB_
RGB_COLORS );
   LPBYTE p = lpvBits+line*(m_nHeight-1);
   int left;
   HRGN hRgn=NULL;
   for( y=0; y<m_nHeight; y++ ){
       left = -1;
       for( x=0; x<m_nWidth; x++, p++ ){
           if( *p ){
               if( left<0 ) left = x;
           } else {
               if( left>=0 ){
                   hRgn = UpdateRgn( hRgn, left, x, y );
                   left = -1;
               }
           }
       }
       if( left>=0 ){
           hRgn = UpdateRgn( hRgn, left, x, y );
       }
       p -= line+m_nWidth;
   }
   GlobalFree( pbmih );
   GlobalFree( lpvBits );
   if( hRgn ){
       SetWindowRgn( hWnd, hRgn, TRUE );
       DeleteObject( hRgn );
   }
   RECT rect;
   GetWindowRect( hWnd, &rect );
   MoveWindow( hWnd, rect.left, rect.top, m_nWidth, m_nHeight, TRUE );
}

HRGN RgnWnd::UpdateRgn( HRGN hRgn, int left, int right, int y )
{
   if( hRgn==NULL ){
       hRgn = CreateRectRgn( left, y, right, y+1 );
   } else {
       HRGN hTmp = CreateRectRgn( left, y, right, y+1 );
       CombineRgn( hRgn, hRgn, hTmp, RGN_OR );
       DeleteObject( hTmp );
   }
   return hRgn;
}
```

合である。

さて，話は少々戻るが，ウィンドウのスタイルである。今回は画像をべったりと全面に貼るため，キャプションバーやフレームは不要である。というか，あるとクライアント領域と非クライアント領域を両方処理しなければならず，面倒である。これはCreateWindowEx () 関数へ渡すスタイルWS_OVERLAPPEDWINDOWを，WS_POPUPにしてしまうことで対処できる。ここで，ウィンドウの形式を説明しておいたほうがいいかもしれない。ウィンドウには以下の3つがある。

1）オーバーラップウィンドウ
2）ポップアップウィンドウ
3）チャイルドウィンドウ

オーバーラップウィンドウは，キャプションバーとフレームを持ち，ほかのウィンドウと重なって（オーバーラップして）表示される。一般的なアプリケーションのメインウィンドウはたいていこれだ。それに対し，ポップアップウィンドウは常に親ウィンドウの手前に表示される。ポップアップメニューやダイアログもこれである。また，チャイルドウィンドウは親に貼り付き，親のクライアント領域内でだけ表示され，親ウィンドウが移動すれば一緒に移動する。ボタンなどのコントロール類がこれに当たる。要は，キャプションバーとフレームがいらなくて，好き勝手に動けるウィンドウを作ろうと思ったら，ポップアップウィンドウにするしかないってことだ（と思う）。

親がいなけりゃ関係ないし。ついでにクライアント領域について。キャプションバーとかフレームとかは非クライアント領域で，それ以外の部分はクライアント領域である。この2つの領域は，同じウィンドウの中であっても，かなり違う振る舞いをする。たとえば左ボタンをクリックした場合，クライアント領域であればWM_LBUTTONDOWNメッセージが送られるが，そうでなければWM_NCLBUTTONDOWNが送られる。引数も異なり，座標はWM_LBUTTONDOWNがクライアント座標であるのに対して，WM_NCLBUTTONDOWNはスクリーン座標である。WM_PAINTとWM_NCPAINTなども同様だ。WM_PAINTメッセージで描き換えられる（描き換えてよい）のはクライアント領域だけで，普通のオーバーラップウィンドウなどでキャプションバーをカスタマイズしようと思ったら，WM_NCPAINTメッセージをトラップしなければならない。

ではポップアップウィンドウはというと，WS_CAPTIONなどを一緒に指定しない限り，ウィンドウ全体がクライアント領域である。つまり，WM_NCPAINTを処理しなくとも，WM_PAINTだけでウィンドウ全体をいじれるわけだ。これにより，ウィンドウ描画部分はリスト3（RgnWindow.cpp抜粋：サンプルRgnWindow2）だけで済む。リージョンを作成したときのビットマップハンドルをメモリデバイスコンテキストに設定し，BitBlt () する。ただし，その前後でBeginPaint () とEndPaint () を呼ぶ必要がある。これは更新の必要のある部分だけをクリッピングしてデバイスコンテキストを取得する関数で，WM_PAINTの処理では常套である。せっかくWM_PAINTメッセージが引数として渡してくれたデバイスコンテキストハンドルが無駄になってしまうが，そこはそれである。

ところで，キャプションバーがなくなってしまったので，ウィンドウの移動は自前でやらねばならないのだろうか？　いやいや，そんなことはない。ここにWM_NCHITTESTというメッセージがある。これは，ウィンドウのどの部分をマウスがポイントしているかを調べるメッセージである。ここで，HTCAPTIONを返してやることで，システムはそこがキャプションバーと勘違いし，マウスドラッグでウィンドウを移動できるようになるのだ。サンプルでは，無条件にこのHTCAPTIONを返し，ウィンドウのどこをドラッグしても移動するようにしている。

さて，もともとシステムメニューもクローズボックスもないウィンドウなので，このままではいよいよ終了方法はAlt+F4しかない。それではあんまりなので，右クリックで［終了］メニューを表示するため，リスト4（RgnWindow.cpp抜粋：サンプルRgnWindow2）のようにした。まずコンストラクタであらかじめメニューを作っておく。その際，IDはMM_QUITという文字定数とした。これは最初に定義しておいたWM_USER+1である。WM_USER以降は，ユーザーがアプリケーションで自由に使ってよいIDとされている。

こうしておいて，WM_NCRBUTTONDOWNメッセージがきたら，TrackPopupMenu () でポップアップメニューを表示する。ここで，「なぜWM_NCRBUTTONDOWNなんだ？　クライアント領域なんだからWM_RBUTTONDOWNじゃないのか？」と思うかもしれない。実はマウス関係のメッセージは，直前にWM_NCHITTESTを呼んでいて，その結果を見ているのだ。つまり，HTCAPTIONが返されるため，キャプションバー上だと思ってWM_NCRBUTTONDOWNメッセージを送ってくるわけだ。

TrackPopupMenu () の引数は次のようになっている。

```
BOOL TrackPopupMenu (
    HMENU hMenu,
    UINT uFlags,
    int x,
    int y,
    int nReserved,
    HWND hWnd,
    CONST RECT *prcRect
);
```

uFlagsはメニューのアラインを指定する。普通基準点がメニューの左上になるので，TPM_LEFTALIGN|TPM_TOPALIGNとなる。xとyはその基準点で，マウスの座標でいいだろう。スクリーン座標で指定するが，先

リスト3

```
void RgnWnd::OnPaint( HDC hDC )
{
   PAINTSTRUCT ps;
   hDC = BeginPaint( m_hWnd, &ps );
   HDC hMemDC = CreateCompatibleDC( NULL );
   SelectObject( hMemDC, m_hBitmap );
   BitBlt( hDC, 0, 0, m_nWidth, m_nHeight, hMemDC, 0, 0, SRCCOPY );
   DeleteDC( hMemDC );
   EndPaint( m_hWnd, &ps );
}
```

リスト4

```
#define MM_QUIT WM_USER+1

RgnWnd::RgnWnd( HINSTANCE hInstance )
{
   …中略…
   m_hMenu = CreatePopupMenu();
   MENUITEMINFO mi;
   mi.cbSize = sizeof(MENUITEMINFO);
   mi.fMask = MIIM_ID|MIIM_TYPE;
   mi.fType = MFT_STRING;
   mi.wID = MM_QUIT;
   mi.dwTypeData = "終了";
   mi.cch = strlen( mi.dwTypeData );
   InsertMenuItem( m_hMenu, 0, TRUE, &mi );
}

RgnWnd::~RgnWnd()
{
   DestroyMenu( m_hMenu );
}

LRESULT RgnWnd::WndProc( HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam )
{
   switch( message ){
   …中略…
   case WM_NCRBUTTONDOWN:
       OnNcRButtonDown( wParam, MAKEPOINTS(lParam) );
       return 0;
   case WM_COMMAND:
       switch( LOWORD(wParam) ){
       case MM_QUIT:
           DestroyWindow( m_hWnd );
           return 0;
       }
       break;
   }
   return DefWindowProc( hWnd, message, wParam, lParam );
}

void RgnWnd::OnNcRButtonDown( int nHittest, POINTS pts )
{
   TrackPopupMenu( m_hMenu, TPM_LEFTALIGN|TPM_TOPALIGN, pts.x, pts.y, 0, m_hWnd,
NULL );
}
```

ほども述べたようにWM_NCRBUTTONDOWNは座標をスクリーン座標で渡すので，そのまま放り込めばよい。hWndは親ウィンドウのハンドルを指定する。メニューが選択された場合，ここで指定されたウィンドウに対し，メッセージが発行される。最後のprcRectは，なにも考えずに普通はNULLだ（説明しにくいし）。

これでもし［終了］が選択されれば，親ウィンドウ，つまりRgnWndにwParamの下位ワードにMM_QUITを入れられてWM_COMMANDが渡される。ここでDestroyWindow（）してやれば，連鎖的にWM_DESTROYが呼ばれ，アプリケーションが終了となる。そうそう，最後にメニューを解放するのを忘れないように。

完成したのが図3だ（サンプルRgnWindow2）。任意の場所をつかんでドラッグすると，ウィンドウを移動させることができる。右クリックすれば，ちゃんとメニューもポップアップする。むーん，これでスキンは完璧だね。リソースの画像を好きなものに差し替えれば，立て看板とかになるぞ。8ビットカラー以外は保証できないけど。そうそう，忘れていたが，なんとなく置物にもなりそうなので，タスクバーに表示されないように，CreateWindowEx（）の際に，拡張スタイルとしてWS_EX_TOOLWINDOWを指定しておいた。あと，常に手前に表示とかしたければ，WS_EX_TOPMOSTを指定してもいいかもしれない（私は「常に手前に表示」は自己主張が強くて嫌いだが）。

メディアを再生してみる

なんとなく勢いに乗って，このままプレイヤーにしてしまおうかと。とはいっても，MP3のデコードをしようとかいうんではなくて，DirectX Media SDK 6.0を利用して再生してしまおうかということである。こいつのDirectShowを使えば，MP3の再生なんて，ちょちょっとファイルを放り込むだけである。ってゆーか，結局ActiveMovieを呼んでいるだけではないかと思われるので，MP3プレイヤーではなくて，メディアプレイヤーができてしまうわけだ。うむ，ケガの功名（なにが？）。

そんなわけで，まず付録CDに収録されているDirectX Media SDKをインストールしてもらいたい。すると，インストールしたフォルダのなかにincludeとかlibとかのフォルダができてるから，それらをVC++のほうのフォルダに上書きコピーしてしまうか，あるいはVisualStudioでそちらにもパスを通しておく。このとき，Media SDKのパスを標準のパスよりも先に持ってくること。いくつかのファイルは新しく更新されているからだ。

では再生方法である。ここもステップを追ってみよう。まずはOLEの初期化。

```
CoInitialize(NULL);
```

こーゆーものだと思っておけばいいんではないだろうか。ついで，GraphBuilderインタフェイスの作成。

```
IGraphBuilder *pigb;
CoCreateInstance(CLSID_FilterGraph, NULL, CLSCTX_
INPROC_SERVER, IID_IGraphBuilder, (void **)&pigb);
```

GraphBuilderをQueryInterfaceして必要なインタフェイスを取得。

```
IMediaControl  *pimc;
pigb->QueryInterface(IID_IMediaControl, (void **)&pimc);
IMediaEventEx *pimex;
pigb->QueryInterface(IID_IMediaEventEx, (void **)&pimex);
IVideoWindow *pivw;
pigb->QueryInterface(IID_IVideoWindow, (void **)&pivw);
IMediaPosition *pimp;
pigb->QueryInterface(IID_IMediaPosition, (void **)&pimp);
```

レンダリング。

```
pigb->RenderFile(wFile, NULL);
```

ここで，wFileはメディアのファイル名だが，Unicodeである。charからUnicodeへの変換は，MultiByteToWideChar（）関数を使う。次いでNotifyメッセージの設定。

```
pimex->SetNotifyWindow((OAHWND)hWnd,
WM_GRAPHNOTIFY, 0);
```

メディアの終了などのメッセージが，ここで指定したウィンドウに，WM_GRAPHNOTIFYとして送られる。WM_GRAPHNOTIFYはWM_USER+2などとして設定しておく。で，再生。

```
pimc->Run();
```

再生終了のWM_GRAPHNOTIFYが来れば，インタフェイスを解放。

```
pivw->put_Visible(OAFALSE);
pivw->Release();
pimc->Release();
pimex->Release();
pimp->Release();
pigb->Release();
```

最後にOLEの終了。

```
CoUninitialize();
```

pivwはVideoWindowのインタフェイスである。ビデオを再生した場合には，パカッとウィンドウが開き，そこでビデオが再生されるわけだが，pivw->put_Visible（OAFALSE）;というのは，そのウィンドウを閉じましょうということだ。このなかでMediaPositionのインタフェイスpimpは使われていないが，

```
REFTIME Duration;
pimp->get_Duration(&Duration);
```

で，メディアの長さ（秒単位）が，

```
REFTIME Current;
pimp->get_CurrentPosition(&Current);
pimp->put_CurrentPosition(Current);
```

で現在の再生位置を取得・設定できる（REFTIMEはdouble）。

いやあ，なんて流れるようで機械的な説明なんだろう。WM_GRAPHNOTIFYメッセージの処理はもちょっと詳しくやると，リスト5のようになる。要はMediaEventExのイベントを拾って，EC_COMPLETEかEC_USERABORTがきてたら終了なわけだ。DirectXのほうの記事も読んでいる人はそうでもないだろうが，突然インタフェイスなどといわれて，ちょっとびびった人もいるだろう。ま，こーゆーもんだと思ってくれ。QueryInterface（）とかRelease（）とか不思議なメソッドがあるが，ちょっと特殊なクラスと思ってとりあえずは差し支えない（のかなぁ）。

なにはともあれ，とりあえずこれくらいでWindows Media Playerが再生できるメディアは再生できてしまう。あと必要なのは，ファイルの受け渡しである。ファイルダイアログを開いて……なんてのより，エクスプローラからドロップして再生，のほうが圧倒的に美しいので，ファイルのドロップをインプリメントしてみよう。とはいえ，受け取りだけならば非常に簡単である。まずは，

```
DragAcceptFiles(hWnd, TRUE);
```

として，hWndで示されるウィンドウがファイルのドロップを許すことを提示する。あるいは，CreateWindowEx（）のコール時に拡張スタイルとして



図3

リスト5

```
LONG evCode;
LONG evParam1, evParam2;
while( SUCCEEDED( pimex->GetEvent( &evCode, &evParam1, &evParam2, 0 ) ) ){
   pimex->FreeEventParams( evCode, evParam1, evParam2 );
   if( (EC_COMPLETE==evCode) || (EC_USERABORT==evCode) ){
       // ここに終了処理を記述する
       break;
   }
}
```

WS_EX_ACCEPTFILESを渡してもよい。そうしたら，あとはメッセージがくるのを待つだけだ。WM_DROPFILESメッセージは，wParamとしてドロップハンドルHDROPが渡されるので，

```
char name[MAX_PATH];
DragQueryFile(hDrop, n, name, MAX_PATH);
```

とすれば，ドロップされたファイルのうち，(0から数えて)n番目のファイル名がnameに格納される。また，

```
DragQueryFile(hDrop, -1, NULL, 0);
```

とすれば，戻り値として一度にドロップされたファイルの数が返る。今回は1曲再生なので，無条件に0番目のファイルだけを拾って再生すればいいだろう。

とりあえずこれで再生可能にはなったものの，再生中はなにかが動いていないとやはりそれっぽくない。まずひとつは再生中の曲名(というかファイル名)でもスクロール表示してやるか，と。あと，どの辺を再生しているか分かるように，つまみみたいなものが動く。こーゆーことさせるとなると，タイマでも仕掛けなけりゃならないけど，それについては以前にやったので，詳しくは述べない。要はSetTimer()して，WM_TIMERメッセージを拾えばいいだけだ。

文字の表示は置いといて，先につまみを考えよう。つまみといえばやはり柿の種……と思ったのだが，なんとなく黄色い玉。普通は左端から右端まで動くわけだが，この〝Oh!Xウィンドウ〟は隙間だらけで，チャイルドウィンドウでは欠けてしまう(ウィンドウのリージョンはそのチャイルドウィンドウにも有効)。なわけで，つまみもポップアップウィンドウ(親は〝Oh!Xウィンドウ〟)。でもって，こいつもCreateEllipticRgn()で作ったリージョンの非矩形ウィンドウ。しかし，そうした場合，面倒なこともある。ポップアップウィンドウは親が移動してもついていかないため，取り残されてしまう。そこで，親が動いたときにくるメッセージWM_MOVEをトラップし，そのたびにつまみも移動させる必要がある。文字の表示部分とあわせて，その部分をリスト6(RgnWindow.cpp抜粋:サンプルXPlay)に示す。

まず文字を表示する領域をInvalidateRect()し，OnPaint()を呼ぶことで，クリアしている。m_hFontはあらかじめ作成しておいたフォントのハンドルである。SetBkMode()で背景を塗りつぶさないモードにしておいて，ExtTextOut()で描画。その際，m_nRollPosによりメッセージの横位置を毎回4ドットずつ動かしている。そのあとはつまみの移動である。タイマは200msごとに設定しているが，つまみはその倍の400msごとに動かすようにしている。メディアの長さと現在の位置の割合からつまみのX座標を算出し，移動させる。ただ，水平に動くだけではつまらないので，適当に式をでっち上げて，うねりながら動くようにした。とまあ，こんなところだ。いよいよもって説明がいい加減になってきたが，まあいつものことだと

リスト6

```
void RgnWnd::OnTimer()
{
   RECT rect = {13,4,204,34};
   HDC hDC = GetDC( m_hWnd );
   InvalidateRect( m_hWnd, &rect, FALSE );
   OnPaint( hDC );
   HFONT hFont = (HFONT)SelectObject( hDC, m_hFont );
   int BkMode = SetBkMode( hDC, TRANSPARENT );
   ExtTextOut( hDC, 16-m_nRollPos, 8, ETO_CLIPPED, &rect, m_RollMes, strlen(m_
RollMes), NULL );
   SelectObject( hDC, hFont );
   SetBkMode( hDC, BkMode );
   ReleaseDC( m_hWnd, hDC );
   m_nRollPos += 4;
   if( m_nRollPos>m_nMesWidth ) m_nRollPos = rect.left-rect.right;
   if( ++m_nCount>=2 ){
       OnMove( 0, 0 );
       m_nCount = 0;
   }
}

void RgnWnd::OnMove( int xPos, int yPos )
{
   if( m_pimp ) m_pimp->get_CurrentPosition( &m_Current );
   double d = m_Current/m_Duration;
   RECT rect;
   GetWindowRect( m_hWnd, &rect );
   MoveWindow( m_hTrack, (int)(rect.left+332* d), (int)(rect.top+88+d* cos(d*
   3.1415*3.3)*40-d*44), 20, 20, TRUE );
}
```

思ってあきらめてくれ。

で，完成したのが図4(サンプルXPlay)。メディアファイルをドロップすると自動的に再生が始まり，左上のオレンジ色のバーに，ファイル名と全体時間がスクロール表示される。表示部分とかつまみの座標が決め打ちになってしまったので，スキンの差し替えの自由度は狭くなってしまった。ま，いいか。

ボタンをつける

まあこれでも一応メディアプレイヤーだといい張れないこともないのだが，せめて再生・停止ボタンくらいは必要かな。当初の計画とは方向性がちょっと違うけど。ということでボタンと，あとつまみをドラッグできるようにしようか。再生や停止させること自体は簡単だ。m_pimc->Run()とかm_pimc->Stop()を呼べばいい。つまみを動かしたときも，m_pimp->put_CurrentPosition()で対応する位置にシークするだけ。となると，問題はユーザーインタフェイス部分である。

ボタンはチャイルドウィンドウなどにせずに，スキン画像に直接描き込んでしまうことにする(図5)。キーボードで操作はできないが，もとよりキーボード操作など想定してはいない。で，WM_NCLBUTTONDOWNでクリックされた座標により振り分ける力技である。ここに関しては，特にソースを示す必要もないだろう。

むしろ面倒なのはつまみである。こちらは真面目にやるしかなさそうだ。WM_LBUTTONDOWNされてからWM_LBUTTONUPされるまでSetCapture()してマウスをキャプチャし，そのあいだにWM_MOUSEMOVEで送られてきたX座標からカレント位置を計算する(リスト7・RgnWindow.cpp抜粋：サンプルXPlay2)。WM_LBUTTONDOWNのメッセージ関数がRgnWnd::OnTrackCapture()，WM_LBUTTONUPがRgnWnd::OnTrackRelease()，WM_MOUSEMOVEがRgnWnd::OnTrackMove()である。注意があるとするならば，トラック中は再生を一時停止しておくことくらいだろうか。OnTrackMove()でちょっとごちゃごちゃやっているxの計算は，つまみのX座標の計算だ。m_TrackPoint.xというのは最初にマウスボタンが押されたときのつまみ内の座標，xPosは現在の座標だから，初めの計算はつまみが動くべきスクリーンX座標である。そこから親ウィンドウのスクリーンX座標を引くわけだから，xはウィンドウ座標となり，つまりつまみの横方向移動量なのである。ただし，この値で直接つまみを動かさず，まずカレント値を計算してから，RgnWnd::OnMove()のほうで移動させている。

とりあえず，図6である(サンプルXPlay2)。なんかこう，本当にこれでいいのか？　というくらいすっ飛ばした説明だが，ここまでちゃんと読んでいれば，これくらいどってことないよな？「ボタン押したら引っこめよ」とか，「MP3のタグをちゃんと読んで曲名表示しろよ」とか，「キャプチャ画面の背景なんかあやしいぞ」とかいろいろあると思うが，一応これはこれで終わり。あとは各自で好きにしてくれい。

ちゃんとしたプレイヤーを目指す

といったところで実験は終わり。これらを踏まえ，さらに当初の要求を満



図4



図5

たすメディアプレイヤーを作る。ここまでやってわかったんだけど，やっぱりユニークな形のウィンドウって，扱いづらいね。ファイルをドロップしようと思っても，隙間からデスクトップに落としちゃったりするし。なんだかんだで，やっぱ矩形ってのはいちばん効率がいいのよ。でもせっかく非矩形ウィンドウの実験もしたことだしってことで，ほんのちょっとだけ，わからない程度に角を落とすことにする。意味があるのかどうかはわからない。

まずはデザイン（図7・設計の落書きより）。で，完成（図8，9・サンプルJukeBox）。あう〜，ごめん。ついいつもの癖でバリバリと書いてたら，ちょっと誌面で解説するには大きすぎるプログラムになってしまった。しかも，例によって行き当たりばったりなので，説明サンプルにしてはソースが汚すぎる。そんなわけで，詳細を知りたい人はCD-ROMに収録してあるソースを見てもらうことにして，ここでは要所要所のポイントを押さえる程度にしておこう。

・リストボックス

プレイリストとストックヤードは，コモンコントロールのリストボックスを使っている。ストックヤードはアルバム用とシングル用があり，アルバム/シングル切り替えタブをクリックすることで表示/非表示を切り替えている。つまり，リストボックスは全部で3つだ。なお，プレイリストとは再生用のリストであり，ストックヤードとはあくまでもCDを置いておく棚である。これらのリストボックス中では，プレイリストファイル（拡張子"m3u"）も展開されずにひとつとして数えられ，プレイリスト単位で操作できる。また，1曲単位でもリストに登録することはできる。

リストボックスは，なんらかの操作が行われると，親ウィンドウにNotificationメッセージを送る。たとえば，項目がダブルクリックされたら，WM_COMMANDメッセージとして，wParamの上位ワードにLBN_DBLCLK，下位ワードにリストボックスのIDが，lParamにリストボックスウィンドウハンドルが入れられて送られてくる。ここで，もし何番目の項目がダブルクリックされたかを知りたければ，

```
SendMessage(hWnd, LB_GETCURSEL, 0, 0);
```

としてリストボックスウィンドウにLB_GETCURSELメッセージを送り（ダブルクリックされた時点で，その項目がカレントセルになっている），その戻り値を取得すればよい。

問題となるのはむしろ外観だろう。標準のリストボックスでは，背景が白，文字が黒となり，せっかくスキンでシックに作ったウィンドウとデザインがあわない。そこで，いくつかの手段を講じる必要がある。まずはWM_CTLCOLORLISTBOXメッセージである。リストボックスは，背景を描画する必要があるときに，親ウィンドウに対してこのメッセージを発行し，返ってきたブラシハンドルを使うことになっている。つまり，このメッセージをハンドルし，ダークグレーのソリッドブラシを返す。ただし，このブラシで描画されるのはリストボックスの中のうち，項目のない部分だけである。項目のある部分の背景もダークグレーにし，そのままでは黒い文字は見えにくいので文字の色も変えようと思ったら，WM_CTLCOLORLISTBOXメッセージが送られてきたときにwParamで渡されるデバイスコンテキストハンドルに，強引に背景とテキストのカラーを選択させてしまうとい

リスト7

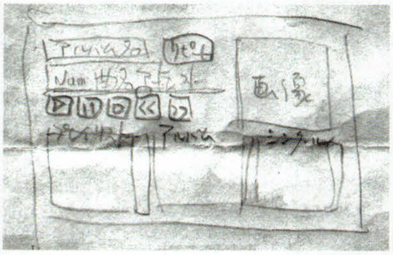
```
void RgnWnd::OnTrackCapture( int xPos, int yPos )
{
    if( m_pimc ){
        m_pimc->Pause();
        m_bTrackCapture = TRUE;
        m_TrackPoint.x = xPos;
        m_TrackPoint.y = yPos;
        SetCapture( m_hTrack );
    }
}

void RgnWnd::OnTrackRelease( int xPos, int yPos )
{
    if( m_pimc ){
        m_bTrackCapture = FALSE;
        ReleaseCapture();
        m_pimp->put_CurrentPosition( m_Current );
        m_pimc->Run();
    }
}

void RgnWnd::OnTrackMove( int xPos, int yPos )
{
    if( m_bTrackCapture ){
        RECT rect;
        GetWindowRect( m_hTrack, &rect );
        int x = rect.left+xPos-m_TrackPoint.x;
        GetWindowRect( m_hWnd, &rect );
        x -= rect.left;
        if( x<0 ) x = 0; else if( x>332 ) x = 332;
        m_Current = x*m_Duration/332.0;
        OnMove( 0, 0 );
    }
}
```



図6


図7

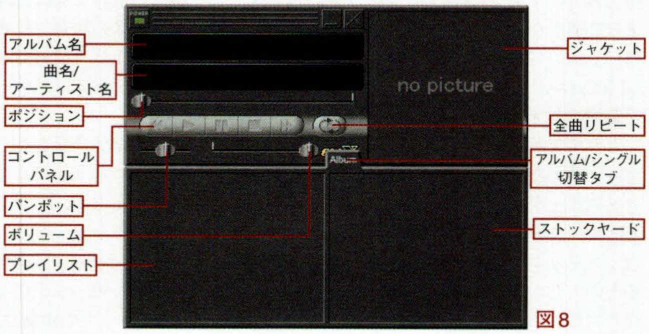


図8



図9

う手がある。

SetBkColor (HDC hDC, COLORREF crColor) ;

SetTextColor (HDC hDC, COLORREF crColor) ;

COLORREF型はRGB () マクロで作ることができる。たとえば，文字を青にしたければ，

SetTextColor (hDC, RGB (0,0,255)) ;

となる。ただし，今回はこれはやっていない。というのは，この方法では結局文字の色は1種類でしか表示できないからだ。ぱっと見てわかるように，リストファイルと単曲のファイル，それに再生中のファイル（ストックヤードではプレイリストにすでに登録されているファイル）を別々の色にしたい。そこで第2段階として，オーナー描画とすることにする。オーナー描画とは，その名のとおり，オーナー（親）がアイテムを描画する方法だ。リストボックスの作成時に，スタイルとしてLBS_OWNERDRAWFIXEDを追加する。オーナー描画のスタイルとしては，もうひとつLBS_OWNERDRAWVARIABLEというのがある。こちらは項目の高さが1つひとつ違う場合のスタイルであるが，今回は色を変えるだけなので，FIXEDのほうで構わない。さて，オーナー描画にした場合，親はいくつかのメッセージを処理しなければならない。ひとつはWM_MEASUREITEMメッセージ。これは，リストボックスが項目の高さを知る必要があるときに，親に問い合わせる。今回は標準的な高さでいいので，無視してそのままDefWindowProc () に渡している。もうひとつは，WM_DRAWITEMメッセージ。これはアイテムが描画される必要があるときに送られるメッセージで，項目1つひとつについて個別に送られてくる。ここで，それぞれに対応した文字の色で描画すればいいわけだ。

なお，リストボックスの項目1つひとつには，ユーザーが自由に利用できる32ビットデータ領域が用意されている。リストボックスに対してLB_SETITEMDATAやLB_GETITEMDATAメッセージを送ることで，ここにデータを設定したり，取得することができる。ここにフラグ（を入れた構造体のポインタ）などを格納しておけば，オーナー描画の情報を別に管理する必要がなくて便利だ。また，オーナー描画リストボックスの場合，作成時にLBS_HASSTRINGSスタイルを指定しない限り，LB_INSERTSTRINGやLB_ADDSTRINGなどの文字列に対するメッセージは，データ領域に対する操作となる。

これで外観についての問題は解消された。ただ，機能的な部分をもうちょっと詰めたい。たとえば，リストボックス内で右クリックすると，べろっとメニューが現れて，選択中のファイルをプレイリストに送るとか，削除するとか，あるいはリストファイルならば内容一覧を表示するとか。あるいは，ストックヤードからプレイリストへドラッグ＆ドロップできるようにするとか。しかし，一般的なリストボックスがNotificationメッセージを送るマウス操作は，左ダブルクリックだけである。シングルクリックや移動は，親からは知る由もない。

そこで第3段階。いちばん最初のサンプルで使ったSetWindowLong () 関数，あれはユーザー定義の32ビット領域だけでなく，ウィンドウプロシージャのアドレスを書き換えてしまうこともできる。インデックスをGWL_WNDPROCとして，新しく作ったプロシージャを放り込む。戻り値は古いプロシージャアドレスなので，新しいプロシージャで処理しないメッセージは古いプロシージャに渡してしまう。こうすれば必要なメッセージだけを横取りできる。これをウィンドウのサブクラス化という。C++言語のサブクラス化と概念は同じだ。ここでLB_RBUTTONDOWNをトラップしてメニューを表示したり，LB_LBUTTONDOWN，LB_MOUSEMOVE，LB_LBUTTONUPの一連のマウス操作で，ドラッグ＆ドロップ操作をインプリメントすることができる。

・ドラッグ＆ドロップ

そのリストボックス間のドラッグ＆ドロップだが，自分（アプリケーション）の同士のやり取りしか考えるつもりがないのであれば，力技で処理することもできる。が，ここはいっちょOLEのドラッグ＆ドロップをインプリメントしてみよう。

ドラッグ＆ドロップを実現するには，ドロップサーバとドロップクライアントが必要である。ドロップサーバとはデータの送り出し側であり，クライアントは受け取り側である。それぞれは少なくとも以下のインタフェイスを持たなければならない。

ドロップサーバ:IDropSource

ドロップクライアント:IDropTarget

さらに，サーバとクライアント間を行き交うデータの橋渡しとして，IDataObjectインタフェイスを持ったオブジェクトが必要である。そんなわけで，まずはこれらのインタフェイスを持つクラスを作らねばならないのだが，ここは少し汎用的にDragDrop.cppおよびDragDrop.hというファイルに独立させておいた。グローバルメモリを一度にひとつしか送れないという制限つきだが，自分だけでしか使わないのなら用は足りるだろう。使い方の流れをリスト8に示す。

間違えてほかのアプリケーションが受け取ってしまったり，自分がほかのアプリケーションのデータを受け取ってしまっても困るので，フォーマット名はユニークなものをつける。また，データのセット時の，CDataObject::SetData () の第3引数だが，これはCDataObjectをhGlobalのオーナーとするかどうかを示す。もっと具体的にいえば，TRUEにするとhGlobalで示したグローバルデータは，用が済めばCDataObjectが解放することを示す。FALSEの場合は，アプリケーション側で解放しなければならない。本来ならば，たとえばドロップしようとしているカーソルが，リストボックスの上端や下端にきたら，スクロールさせるといった処理が必要だが，今回は省略した。

リスト8

```
// それぞれのインタフェイスを持ったクラスのインスタンス
CDropSource dropSource;
CDataObject dataObject;
CDropTarget dropTarget;

// OLEの初期化
OleInitialize( NULL );

// データのフォーマット名を登録
CLIPFORMAT cfFormat = RegisterClipboardFormat( "FooFormat" );

// ターゲット側:受け取り可能なフォーマットの設定
FORMATETC formatetc;
formatetc.cfFormat = cfFormat;
formatetc.ptd = NULL;
formatetc.dwAspect = DVASPECT_CONTENT;
formatetc.lindex = -1;
formatetc.tymed = TYMED_HGLOBAL;
dropTarget.AcceptFormatEtc( &formatetc );

// ターゲット側:ドロップされたときのコールバック
dropTarget.SetCallback( DropCallback );

// ターゲット側:ドロップの許可
RegisterDragDrop( hWnd, &m_dropTarget );

// サーバー側:データのセットとドラッグ開始
STGMEDIUM stgmedium;
stgmedium.tymed = TYMED_HGLOBAL;
stgmedium.hGlobal = hGlobal;
stgmedium.pUnkForRelease = NULL;
dataObject.SetData( &formatetc, &stgmedium, FALSE );
DWORD dwEffect;
if( DoDragDrop( &dataObject, &dropSource, DROPEFFECT_COPY|DROPEFFECT_MOVE,
   &dwEffect )==DRAGDROP_S_DROP ){
   // ドロップ成功
}

// ターゲット側:ドロップの禁止
RevokeDragDrop( hWnd );

// OLEの終了
OleUninitialize();


// ターゲット側:コールバック関数
LRESULT DropCallback( LPDATAOBJECT pDataObject, DWORD grfKeyState, POINTL pt,
   LPDWORD pdwEffect )
{
   STGMEDIUM stgmedium;
   if( pDataObject->GetData( &formatetc, &stgmedium )==S_OK ){
       if( stgmedium.tymed==TYMED_HGLOBAL ){
           HGLOBAL hGlobal = stgmedium.hGlobal;
           // 処理を行う
       }
   }
   return S_OK;
}
```

また，いろいろと思うところがあって，データとしてリストの項目ではなく，リストボックスのウィンドウハンドルだけしか送っていない。ウィンドウハンドルひとつ送るのに，わざわざグローバルメモリを確保するのも馬鹿らしいので，グローバルメモリハンドルと偽ってメモリハンドルを渡している。受け取り側もそれを知っているので，データのオーナーをCDataObjectにしなければ特に問題はないはずだ。

その他の問題としては，エクスプローラなどからのファイルのドロップが受けられなくなる。これはちょっと困った。ちゃんとファイルも受け取るように，その辺を自前で処理するのも面倒だ。どうせ相手も自分なわけだし，ということで，ドラッグを開始する直前にターゲット側のドロップを許可し，ドラッグが終了したらドロップを禁止するという，コスい手を使った。こうしておけば，その他のときにはファイルのドロップを受けられる。なんかこう，OLEのメリットをことごとく叩き潰しているような気がするのだが，深いことは気にしないことにしよう。

・ジャケットの表示

ジャケットの領域は160×160ドットで，リストファイル名または単曲のファイル名のベース名で，拡張子が〝jpg″，〝gif″，〝bmp″の順に探した画像を表示する。フォルダはファイルと同じところを探し，なければ，ジャケットフォルダに設定したフォルダ内を検索する。なお，画像サイズが160×160ドットでなくても，拡大や縮小は行わず，真ん中に表示するようにしている。

画像ファイルのロードには，某フリーソフトのプラグインなんかを使うことも考えられるが，ここはDirectShowついでに，これまたDirect Media SDKに入っているDirectX Transformを使うことにする。画像に対してエフェクトを行うことを目的としたAPIだが，それをロードのためだけに使うなんて，ちょっと贅沢な気分だろう？（無駄ともいう）

手順としてはリスト9のようになる。

特に注意が必要な部分はないと思うが，強いていうならファイル名がUnicodeである点くらいだろう。それほど手軽というわけではないが，JPGだろうがGIFだろうがBMPだろうがファイル名を渡すだけで同じ手順でロードできる。あとはデバイスコンテキストを取得して，ウィンドウのデバイスコンテキストにBitBlt ()するだけだ。

・MP3タグ

さすがにファイル名しか表示しないのでは不便なので，MP3のタグを読むようにしてある。手元に資料がなかったので（探してもいないが），適当にファイルを覗いてみたら，ファイルの最後の128バイトがそうであることがわかった。最初の3バイトに〝TAG″の識別子があり，続いて30バイトずつ曲名，アーティスト名，アルバム名が入っている。とりあえずそれ以上は必要ないので放ってあるが，もしタグがあれば，それをもとにリストに表示，およびロールメッセージを作成している。

・終了処理

終了時には，そのときの状態をファイルなりレジストリなりに保存しておく必要があるだろう。ウィンドウの座標や現在再生中のファイルの番号などもだ。そうすれば，次に起動したときに，前回と同じ状態になり，続きを再生できる。

レジストリやファイルについてはいまさら説明の必要もないと思うが，ちょっと注意が必要なのは，状態保存のタイミングである。たとえばOnDestroy ()。それはそれで間違いではないが，それだけでは不十分である。というのは，アプリケーションを終了させないままWindowsを終了した場合，実はアプリケーションのウィンドウが破棄されないだけでなく，メモリからも解放されないのだ。そのため，OnDestroy ()は呼ばれず，次回起動時に以前の状態に戻らない。どうせシステム全体が終了するのだから，ウィンドウを1つひとつ破棄する必要はないということだろうが，プログラマ側から見れば結構面倒な話である。実際，筆者が使っていたMP3プレイヤーは，Windowsごと終了すると以前の状態が残っていないので，この落とし穴にはまっていると思われる。

ではどうするか。Windowsはユーザーから終了要求を受けると，稼働しているアプリケーションに対してWM_QUERYENDSESSIONメッセージをブロードキャストする。だが，それだけではまだシャットダウンされない。そのメッセージに対してすべてのアプリケーションがTRUEを返せばいいが，FALSEを返すアプリケーションがひとつでもあると，終了処理は中止される。たとえば，更新を保存していないアプリケーションがあったとしよう。普通はその状態でWindowsを終了しようとすると，保存を促すダイアログが表示される。そのダイアログはWM_QUERYENDSESSIONメッセージを受けて表示されたものである。そこで「はい」を選択して保存するなり「いいえ」を選択すれば，そのアプリケーションはWM_QUERYENDSESSIONメッセージの戻り値としてTRUEを返す。ところが，「キャンセル」を選ぶと，FALSEを返し，Windowsは終了しないというわけだ。さらに，すべてのアプリケーションがWM_QUERYENDSESSIONメッセージを処理すると，今度はWM_ENDSESSIONが送られてくる。これはwParamにBOOL値が入っており，すべてのアプリケーションがWindowsの終了を許可した場合にはTRUE，ひとつでも拒否した場合はFALSEとなっている。つまり，最終的な終了処理はwParamにTRUEが入ったWM_ENDSESSIONメッセージを受け取ったときであり，ここでもOnDestroy ()と同様の処理が必要になってくる。

Windows Media Audio

Windows Media Playerの6.4からは，期待のWindows Media Audioの再生も可能になっている。これはコードネームMS Audioと呼ばれていたフォーマットだ。同程度の音質ならば，MP3の半分程度まで圧縮できる。Windows Media Player 6.4がインストールされていれば，今回のJukeBoxでもこれが再生可能である。ただし，曲名などの情報は表示されない。Windows Media Audio SDKを使えば，その辺りの情報を取得することも可能なのだが，今回は時間切れということで。

バグはいっぱいある。挙動も不審であったりする。ソースには例によってコメントがない。仕様は（U）氏にはなにか不評である。が，とりあえず使えるものはできたはずだ。現にこの原稿も，このJukeBoxでMP3をBGMに書いているが，致命的な問題は起きていない（だって流しっ放しにしてるだけだもんな）。ひょっとしたら，この本が出ているころには筆者のホームページにWindows Media Audioに正式対応したバージョンが上がっているかもしれないが，いまはこれくらいで勘弁して。そうそう，JukeBoxにはヘルプもつけとくから，それをちゃんと読んでから使ってね。

リスト9

```
// インタフェイスのポインタ
IDXSurfaceFactory *pSurfFact;
IDXSurface *pDXImageSurf;

// OLEの初期化
CoInitialize( NULL );

// IDXSurfaceFactoryインタフェイスの作成
CoCreateInstance( CLSID_DXTransformFactory, NULL, CLSCTX_INPROC,
   IID_IDXSurfaceFactory, (void **)&pSurfFact );

// 画像からサーフェスを作成
pSurfFact->LoadImage( szWFileName, NULL, NULL,
   &DDPF_PMARGB32, IID_IDXSurface, (void**)&pDXImageSurf );

// 画像のサイズを取得
CDXDBnds bounds;
pDXImageSurf->GetBounds( &bounds );
width = bounds.Width();
height = bounds.Height();

// デバイスコンテキストの取得
IDXDCLock *pDCLock;
pDXImageSurf->LockSurfaceDC( NULL, 3000, 0, &pDCLock );
HDC hdcSurf = pDCLock->GetDC();

// デバイスコンテキストの解放
pDCLock->Release();

// インタフェイスの解放
pDXImageSurf->Release();
pSurfFact->Release();

// OLEの終了
CoUninitialize();
```

H"・文字電話でWebを見よう!

大和 哲　Yamato Satoshi

文字電話というとちょっと特殊ですが，いま流行のH"など，Web接続機能のないただのメール端末でもWebページを見てみたい……という奇特な方には朗報です。Webページをメールにして転送するCGIスクリプトを作ってみましょう。

最近は携帯電話をはじめとした携帯機器でWebを見られるものがかなりいろいろあります。たとえば，今号のもうひとつの記事「iモード用のCGIを作ろう！」のほうで取り上げている，NTT ドコモの「iモード」などがそうです。このiモードサービスは，iモード対応携帯電話に電話をかける機能以外に「iモードメニュー」というものがあって，この中の「Internet接続」という項目を選んで「http://～」とURLを記入すると，そのURLに存在するWebページデータの内容が電話の液晶画面に表示される，というものです(かなり大雑把な説明ですけどね)。

さて，ところで，筆者は実は最近，結局iモード携帯電話も買ってしまったのですが，携帯電話は高いので，iモードを買う以前は「文字電話」端末を使っておりました。これはPHSの事業者であるDDI Pocketのメールサービス「Pメール」「PメールDX」専用携帯端末でして，全角1000文字以内の日本語のメールの送受信が可能で，そして，○×@pdx.ne.jpというアドレスを取得することでインターネットメールの送受信も可能となるものでした。安いんですよね，これが，基本料金は月980円，1回送受信に10円ですから。

ところが人間というのは欲が出てくるもので，どうしてもiモードのように，メールだけでなくWebも見たいなぁ，などと思ってしまうのですね。そこででっちあげたのが，今回サンプルプログラムとして紹介する「webmail.pl」です。このプログラムを使うと，メールしか読み書きできない文字電話でWebページを，無理やりですが，読むことができます。



サーバ環境としては，UNIX系のOSでSMTPサーバにsendmailを使っていて，.forwardでメールの転送ができる，Perlのスクリプトをユーザー環境で実行できる，などの環境が必要になります(ちなみに，筆者がホームページを置いているサーバはRedhatLinux，ユーザープログラム実行可，telnet可です。まぁ，そういう自由度のホスティングを探して入ったわけですが……)。それから夏号のサンプルプログラムと同様CPANモジュールが必要になります。本ページの最後に記載されているように管理者の方に必要なモジュールをインストールしてもらうか，夏号でGD.pmを使用したのと同じようにユーザーレベルでインストールを行って，スクリプトのほうからそのモジュールにPATHを通して使ってください(詳細はOh!X夏号「Webページでグラフを表示しようII」の記事を参照してください)。

どのようにしてWebを見るのか

さて，どのようにして，メール端末でWebを閲覧するのか。仕組みは簡単です。

インターネット上のサーバに筆者の作ったプログラム「webmail.pl」があります。これは，メールが着信するとプログラムが起動し，ほかのインターネット上のWebコンテンツを取ります。そして，この内容の文字部分をメールとして，再び文字電話に送るようになっています。つまり，文字電話側からすると，

- 文字電話でURLの書かれたメールを書く
- メールを「インターネットメール」であるアドレスに送る
- しばらく待つ
- サーバからURLで書かれていたホームページが返ってくる

という感じになります。ちなみに作ったスクリプトを使ってみるとおおよそ30秒から1分くらいで1ページが見られるという感じです。

ちなみに，iモード携帯電話がどのようにWebコンテンツ取得を行っているかというと，こちらは，まず，iモード端末が携帯電話の回線を使ってNTT ドコモのセンターに接続されます。で，iモードというのはInternet接続以外にも，バンキングや航空券の予約など「iモードサービス」と呼ばれるサービスがあって(というか，iモードとしてはこちらが本筋のサービスなのでしょうけど)，それぞれのサービスの提供者がいるわけです。NTT ドコモは携帯端末からのサービスのリクエストを受けると，それをサービスの提供者に送信し，提供者から返ってきた答えを再びリクエスト元のiモード携帯端末に送信します。で，Internet接続リクエストの場合は，NTT ドコモがここでリクエストをiモードサービス提供者ではなく，NTT ドコモにある，インターネットに接続されたProxy (代理) サーバにリクエストを送ります。Webサーバがそれに対するレスポンスを送り，Proxy，携帯端末，とレスポンスが返ってきて携帯端末上にWebページの内容が表示される，とこういうことになっているわけなのですね。

ですので，ドコモのProxyサーバと同じように，インターネット上で文字電話からのリクエストのメールをどこかのサーバで受け付けて，それがインターネット上のコンテンツを取ってきて，端末にメールしてくれれば，なんだ，iモードの代わりになるじゃんか！　という思いつきで作ってしまったものなのですね(でも，よく考えたら，文字電話も，占いやら，伝言板やらのサービスがあるんだから，はじめからDDIポケットさえ，もうちょっ

と気合を入れてWebにも対応してくれてもよかったんじゃないか!?　という気も……ま，いってもせんないことではあります。はい)。

ともあれ，せっかくですから，この私がでっちあげた「webmail.pl」の作りなどをご説明していきましょう。このプログラムは単にPerlで書かれたスクリプトで，およそ100行弱。内容はざっとこんな感じになっています。

・届いたメールの解釈部分
・解釈に応じてインターネット上のWebサーバのhtmlコンテンツを取得する部分
・取得したhtmlコンテンツをメールに変換して，送信者に返信する部分

Webコンテンツを取得するには

Oh!X 復刊1号の記事「あなたの知らないWebサーバの世界」のサンプルプログラムでは，CGIでほかのWebサーバのページを，Socketライブラリを使って自分でWebサーバのポート80を叩くプログラムを作って，獲得してみました。このSocketを使って，自力で80番ポートを使ってWebサーバと接続し，コマンドを送信して，コンテンツを取得……という方法を今回も使ってもよかった……今回のプログラム「webmail.pl」では，それよりもっと楽な方法を使っています。というのも，Socketを使って自力でWebコンテンツを取得するようなプログラムを書くと，セキュリティなども考慮して自力で例外処理などをせねばならず結構インプリメントが面倒な個所が多いからなのです。たとえば，タイムアウトの処理。まじめにやったら，forkでプロセスを分けて，ポートを叩いている部分とそれが長い間止まっていないか監視する部分を作って……とちょっとうんざりします。

さて，そこで，今回はCPANモジュールを使用することにしました。

前回のCGIプログラミングの記事を覚えていらっしゃいますでしょうか？　アクセスカウンタグラフの記事です。この記事中で，折れ線グラフを表示するために，CPANという，総合Perlライブラリで「GD.pm」というグラフィックモジュールがあるので，これを使ってイメージファイルの操作を行いました。

このCPANライブラリにはグラフィックから，ネットワーク，文字列操作，データベース，ファイル操作など実にさまざまなモジュールがあります。実は，インターネット上のWebコンテンツを取得するためのモジュールというのも存在しているのです。このモジュールは「LWP (libwww-perl)」という名前で，User-Agent名の設定，Proxyの設定，Webロボットルールの遵守設定，URLの解釈，そして，Webコンテンツの取得などのメソッドがびっしりと詰め込まれていて，数行スクリプトを書くだけでこれらを使ったプログラミングができてしまいます。今回はプログラムを簡単に作るために以下のライブラリモジュールを使用しています。

・このプログラムに使用したライブラリモジュール

```
LWP::UserAgent;
HTTP::Request;
HTTP::Response;
Jcode
```

LWP::UserAgentはWebブラウザなど，ユーザー側のインターネット閲覧ツールを作るときに使われるモジュール。これを使うと簡単にほかのWebサーバの内容を獲得できます。もちろん，タイムアウトの監視も関数をひとつ呼び出すだけ，User-Agentも名乗ってくれる，Proxyサーバにもコード1行で対応できる，ことWebアクセスに関してはいたれりつくせりのモジュールです。ちなみに，これによく似たモジュールとして「LWP::RobotUA」というのも存在しまして，これは機能はほぼ同じなのですが，検索エンジンのように自動的にインターネット上のコンテンツを定期的に拾いに行くようなWebロボットを作成する場合に使われます。というのも，こちらは「robot.txt」など，ロボット用に作られたルールの解釈も行ってWebコンテンツの取得などを行うようになっているのです。この記事を読んでWebロボットなどを作ろうと思った方はそちらを使うようにしてくださいね(LWP::RobotUAはLWP::UserAgentのサブクラスですので，メソッド名などはこの記事も参考になるだろう……とは思いますけど)。

HTTP::RequestはHTTPというスキームを使ってインターネット上のWebサーバをアクセスするためのオブジェクト，HTTP::Responseはアクセスした際のレスポンスが入るオブジェクトです。これらのごく簡単な使い方はこんな感じになります。$urlという変数がアクセスするWebコンテンツのURIだと思ってください。

```
my $ua = new LWP::UserAgent("エージェント名","管理者のメールアドレス");
my $request = new HTTP::Request('GET',$url);
my $response = $ua->request($request);
```

3行目を見てください。Webコンテンツのアクセスの要はここです。

```
$ua->request($reqest)
```

つまり，あなたが作った「ユーザーエージェント」にあらかじめ作っておいた「リクエスト」の内容でリクエストさせる，つまり動作させる，という

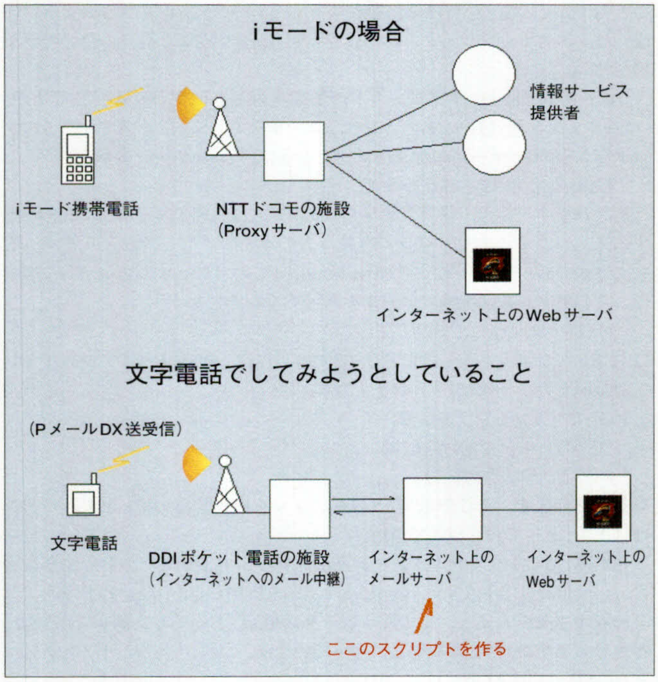


図1　サーバの概略

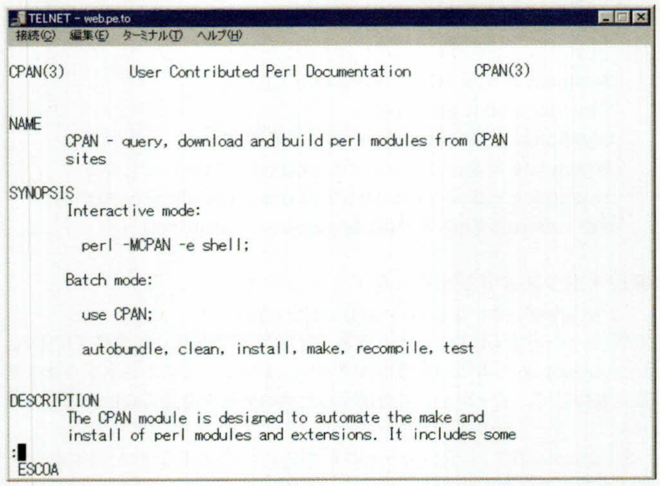


図2　telnetでWebサーバにアクセスできれば，「perldoc モジュール名」でインストールされているPerlモジュールのヘルプが見られるかもしれない。試してみよう(画面は某ホスティングサーバで見てみたPerldoc CPANの結果)

ことです。その前の2行はそれぞれユーザーエージェントオブジェクトの作成，リクエストオブジェクトの作成ですから，それぞれ3行目のための下準備ですね。

LWP::Useragentオブジェクトの初期化のときに使われる引数ですが，「エージェント名」はその名のとおり，ユーザーエージェント名です。このプログラムがWebサーバにアクセスしたときに，UserAgent変数が，

```
UserAgent: エージェント名
```

と渡されます。また，管理者のメールアドレスは，このエージェントを管理する人，つまりこのプログラムを使う方のメールアドレスが入ります。もし，ほかのWebサーバが，このwebmail.plでのアクセスに文句がある場合は，このアドレスに抗議のメールを送ってくるでしょう。

```
HTTP::Request('GET',$url)
```

は「URLのアドレスをGETする」ですね。復刊1号でのCGIプログラミングで説明したようにHTTPのリクエストは，

```
GET　コンテンツを取得
HEAD　ヘッダだけを取得
POST　データの転送
```

などがあります。ここではWebコンテンツを取得するので，1番目の引数は「GET」としておくわけですね。

ちなみにPerlのドキュメントを表示させるperldocコマンドが使える場合，「perldoc LWP::UserAgent」として，LWP::UserAgentのドキュメントを表示させてみると，このモジュールの概要（というか，簡単な使い方，だと思いますが）はこんな感じになっています。

```
LWP::UserAgent(User Contributed Perl
DocumentatiLWP::UserAgent(3)

概要
$ua = new LWP::UserAgent;

$request = new HTTP::Request('GET',
'file://localhost/etc/motd');
$response = $ua->request($request);# or
$response = $ua->request($request, '/tmp/sss'); # or
$response = $ua->request($request, ¥&callback, 4096);
sub callback { my($data, $response, $protocol) = @_; .... }
```

●レスポンスオブジェクトをチェックする

```
my $response = $ua->request($request);
```

でWebページをGETすると，それが正常終了であれ，異常終了であれ，$responseにもろもろの情報が入ります。よって，ここからさまざまな情報を取得して，メールとして発送するためのデータを作らなければなりません。

$response（HTTP::Responseのオブジェクト）の主なメソッドには以下のようなものがあります。

- $r = HTTP::Response->new($rc, [$msg, [$header,[$content]]])

  HTTP::Responseオブジェクトのコンストラクタです。応答コード$rc，オプショナルメッセージ$msg，ヘッダ$header，コンテント$contentを初期設定できます。

- $r->is_success

  引数なしでHTTPリクエストが成功したのかどうかを参照できます。成功した場合，値は真です。

- $r->is_error

  引数なしでHTTPリクエストが失敗したのかどうかを参照できます。成功した場合，値は真です。

- $r->is_redirect

  HTTPリクエストの結果がリダイレクションであったかどうかを参照できます。リダイレクションであった場合，値は真です。

- $r->content

  取得したHTMLテキストが入っています。

- $r->error_as_HTML

  HTTPリクエストに失敗した場合，エラーメッセージをHTMLの形で表示します。

- $r->code([$code])

  引数なしでは，現在の応答コードが参照できます。

- $r->message([$message])

  引数なしでは，現在のHTTPリクエストのメッセージを参照できます。

- $r->base

  現在のベースURLを参照できます。

また，HTTP::Response はHTTP::Messageのサブクラスですので，これらのメソッドはすべて継承します。

たとえば，

```
header();
push_header();
remove_header();
headers_as_string();
```

などを使うことができます。詳細はperldocの使える環境で「perldoc HTTP::Message」を参照してください。

さて，そのperldocによると，このHTTP::Responseの代表的な使い方は以下のようになるそうです。以下はGETリクエストを行い，リクエストが成功だったら，そのHTMLコンテンツをそのまま表示し，エラーだった場合にはエラー内容をHTMLで表示するサンプルです（コメントは筆者によるもの）。

```
#リクエストを発行
#レスポンスが変数に入る
$response = $ua->request($request)

#リクエストは成功か?
if ($response->is_success) {
    #成功した場合の処理
    #(HTMLでの表示)
    print $response->content;
} else {
    #失敗した場合の処理
    #(HTMLでの表示)
    print $response->error_as_HTML;
}
```

今回のwebmail.plの場合はHTMLではなく，プレーンテキストで書かなくてはなりません。ですので，このリストとは逆に成功，失敗，それぞれ「コンテンツからHTMLタグをすべて取り去って（つまりプレーンテキストにして）$textという変数に入れる」「エラーコードを取得して，エラーメッセージを作成して$textという変数に入れる」という処理をしています。具体的にはサンプルリスト63行目から77行目あたりで，

```
if($response->is_success){
  $text=$response->content;
  $text = Jcode->new($text)->h2z->euc;
  $text =~ s/<[^>]*>//g;
  $text =~ s/&lt;/</g;
  $text =~ s/&gt;/>/g;
 } else {
  my $i = $response->code();
  $text = "Webページを取得できませんでした。¥n";
  $text .= "エラーコード:$i¥n";
  $text .= "by mojiweb¥n";
}
```

のように処理をしています。ちなみに，日本語のWebページでは，いろいろな漢字コードが使われていますが，処理を簡単にするためにJcodeを使っていったん，EUCコードに変換しています。リスト中の，

```
$text = Jcode->new($text)->h2z->euc;
```

がそれにあたります。

シフトJISコードやJISコードにはコード中に半角英数字で使用されてい

るキャラクタコードと重複する部分があり，jperlのようにマルチバイトキャラクタのことをはじめから考慮している処理系でないと，そのまま文字列を処理しようとすると誤った処理を行ってしまうことがあります。たとえば，上のリストでも，HTMLタグをすべて取り除くために，

```
$text =~ s/<[^>]*>//g;
```

と，「<」「>」で囲われた部分をすべて削除していますが，JISコードにはこの「<」「>」と同じ$3C，$3Eという値を使う文字コードが大量に存在するため，GETで取得したコンテンツにそのまま，この変換をかけてしまうと，本来HTMLタグとは関係のない日本語の文章中の文字が削られてしまい，結果として文字化けを起こしてしまいます。

EUCコードは，シフトJISと文字コード体系は似ていますが（簡単にいうと，漢字などの2バイトキャラクタでは，シフトJISの文字コードと論理OR$8000を取るとEUCになります），2バイトキャラクタはすべてASCIIの半角英数字のキャラクタの範囲外である$80以上の範囲に収まっており，2バイトキャラクタ（つまり，全角文字）の文字操作さえ行わなければ，英語版の処理系をそのまま使うことができます。今回は半角英数字記号である「<」「>」の処理を行うだけですので，この方法を取っています。

なお，今回はプログラム中でも，

```
use Jcode;
$text = Jcode->new($text)->h2z->euc;
```

としていることからもわかるようにいままで使っていた日本語文字コードをPerlで扱うためにjcode.plを使っていましたが，今回はこれのモジュール版であるJcode.pmを使用しています。今回はLWPなどのCPANモジュールを使用することが前提，というのもありますが，今回使うと便利だろうなと思った機能がいくつか追加されているためです（webmail.plでは結局使わなかったのですが……）。

ひとつはMIMEヘッダ関連機能。たとえば，日本語をヘッダに使用した場合，ヘッダ中には日本語の文字（に限らず8ビットでないと表現できない$80以上のコードを含むすべての文字列）MIMEエンコード，というデータの変換作業を行って表現しなければいけないのですが，Jcode.pmを使用すると，たった1行でこのエンコードを行うことができます。

```
メールテスト
```

という文字列をMIMEエンコードしたければ，

```
$subject = jcode( "メールテスト")->mime_encode;
```

とするだけで，$subjectには，

```
=?iso-2022-jp?B?GyRCJWEhPCVrJUYlOSVIGyhC?=
```

とエンコードされた文字列が入るのです。

また，もうひとつ，このJcode.pmはまだjcode.plが対応していないUTF-8にも対応しているということが挙げられます。まだ，UTF-8で書かれたWebページはそれほど多くありませんが，筆者が知る限りでは「Namo Webエディタ」のようにUTF-8で書かれたページデータを作るソフトもありますので，Jcode.pmを使用することで対応しておいて損はないはずだ，という判断です。Jcode.pmについての解説は，

http://openlab.ring.gr.jp/Jcode/index-j.html

にあります（日本語です）ので，参照してみてください。

なお，余談ですが，今回のwebmail.plではHTML中のプレーンテキスト部分を抜き出して送信するだけのことしか行っていませんが，たとえば，逆にテキストでなくハイパーリンクだけを抽出する，あるいは特定のタグだけを抽出するなどのプログラムを作りたくなる場合もあるでしょう。CPANライブラリには「HTML」で始まるHTML加工用のさまざまなモジュールがありますので，そのような加工をしたい場合には，まずCPANのWebサイトをご覧になることをおすすめします。たとえば，先ほどの，HTML文中のリンクだけを抽出するのであれば，

```
HTML::LinkExtor;
```

という専用のモジュールがあります。これを使うと，便利ですよ。

メールの解釈，メールの構造

さて，話はプログラムでいうと少し前の行に戻ります。今回のプログラムwebmail.plでは，文字電話から送られてきたメールの題名と送信者のデータを参照し，メールの題名に書かれている文字列をURLとみなして，そのWebページの内容を送信者に返します。

リスト1　H"からのメール（文字だけのメール）

```
-------------------
From skomuja@pdx.ne.jp Wed Jan 19 19:40:11 2000
Received: from m4.gw.shippo.co.jp (m4.gw.shippo.co.jp [10.0.50.74])
   by t22.shippo.co.jp (8.8.8/8.8.8) with ESMTP id TAA09708
   for <mailtest@t22.shippo.co.jp>; Wed, 19 Jan 2000 19:40:11 +0900 (JST)
   (envelope-from skomuja@pdx.ne.jp)
From: skomuja@pdx.ne.jp
Message-Id: <200001191044.TAA10712@pdxemo2.pdx.ne.jp>
Date: Wed, 19 Jan 2000 19:40:30 +0900 (JST)
Subject: =?ISO-2022-JP?B?GyRCJEYkORsoQg==?=
To: mailtest@t22.shippo.co.jp
MIME-Version: 1.0
Content-Type: Text/Plain; charset=iso-2022-jp
Status: RO

てす
```

リスト2　H"からのメール（画像付き）

```
-------------------
Received: from pdxout1.pdx.ne.jp (pdxout1.pdx.ne.jp [210.168.199.48]) by t22.
shippo.co.jp (8.9.3/3.7W) with ESMTP id LAA03216
for <webmail@shippo.co.jp>; Mon, 20 Dec 1999 11:12:14 +0900
From: skomuja@pdx.ne.jp
Received: from pdxemo1.pdx.ne.jp by pdxout1.pdx.ne.jp (8.8.6 (PHNE_17190)/8.8.6)
with ESMTP id MAA06399
for <webmail@shippo.co.jp>; Mon, 20 Dec 1999 12:52:03 +0900 (JST)
Received: (from root@localhost)
by pdx.ne.jp (8.8.6 (PHNE_17190)/8.8.6) id MAA14328
for webmail@shippo.co.jp; Mon, 20 Dec 1999 12:55:15 +0900 (JST)
Message-Id: <199912200355.MAA14328@pdx.ne.jp>
Date: Mon, 20 Dec 1999 12:51:48 +0900 (JST)
To: webmail@shippo.co.jp
MIME-Version: 1.0
Content-Type: Multipart/Mixed; Boundary="?000"
X-UIDL: 1GIe9?f7!!,ad!!UM1e9

--?000
Content-Type: Text/Plain;charset=us-ascii
   メールテスト

--?000
Content-Type: Application/Octet-stream;Name="memo.bmp"
Content-Transfer-Encoding: Base64
X-PmailDX-Info: DIV6

Qk1uDAAAAAAAAD4AAAAoAAAAwAAAAIIAAAABAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAP///wD///////////////////////////////////////////////////////////
(中略)
//////////////////////////////////////////////////////////////////8=
--?000--
```

プログラム中の17行目から38行目あたりまでが，メール中からヘッダの内容を取り出して，各連想配列にヘッダのタグと内容を格納する部分です。

一般にインターネットメールはRFC822，また2バイトコードを使用するヘッダや添付ファイルなどのMIMEメールに関してはRFC2045～2048などで定義されています。それによると（必要最低限のごくかいつまんだ説明ですが），インターネットメールは必ず，

```
ヘッダ
¥n
ボディ
```

あるいはマルチパートのメールではヘッダ中のContentTypeヘッダフィールド中にBoundary=″?000″と指定された境界を使って，

```
ヘッダ
¥n
--?000
コンテント1
¥n
--?000
コンテント2
¥n
 ⋮
```

と境界と改行文字を使って，１つひとつのパートが分けられるようになっています。ヘッダは必ずメールの先頭にあるので，メールを初めから読んでいって，改行文字が出現するまでそれぞれのヘッダフィールドの内容を連想配列に，

```
$header{Subject}="題名";
$header{From}="送信者";
```

というように入れていっているのです。これで解釈したメールの題名をLWPモジュールで実際にWebページをGETしている部分に「GETすべきURL」として，送信者欄に書かれていたメールアドレスをメールの送信部分に「メールの送信先」として，変数に入れて渡しています。

ちなみに，このプログラムは文字電話，H"専用なわけですが，H"から送られたメールを除いてみると内容はリスト１のようになっていました。ご参考までに。

メールの送信

さて，CGIをはじめとした，Perlスクリプトでメールを送信するにはいくつかの方法があります。たとえば，SMTPサーバにSocketを使って接続して25番ポート（SMTPポート）を直接叩いて送信する方法。あるいはCPANのSMTPモジュールを使う方法。ここでは簡単に「sendmail」というプログラムを使います。

これはUNIX系のOSにはだいたい標準で添付されているメール関連の機能を司どるプログラムで，コマンドライン上からメールを送ることもできます。もし，telnetが使えるサーバならば，sendmail 宛先と入れて，そのあとお好きな内容を書き込んでください。

```
>sendmail foo@hoge.co.jp
test
```

最後に「ctrl」+「D」を押すと，メールを送信することができます。

ところで，PerlからUNIXのコマンドを呼び出すにはいくつかの方法があります（というか，どんなことにもいくつものやり方があるのがPerlなんですが…）。たとえば，簡単なやり方だと，

```
system('コマンド');
```

あるいは，出力結果がほしいなら，

```
$result = `コマンド`;
```

なんていうのもあるのですが，ここではファイルのオープンクローズと同じようにopen，closeで使っています。open，closeでは，たとえば，

```
open(ハンドル,"|コマンド") ;
```

でそのコマンドへ結果を渡すことができます。ちなみに，

```
open(ハンドル,">ファイル名");
```

でファイルへの書き込み（リダイレクト）を行います。つまりUNIXやMS-DOSのコマンドラインのパイプライン（|），リダイレクション（>）と同じなのですね。さらに，

```
open(ハンドル,"<ファイル名");
```

もファイルからの入力です。

で，sendmailを使って，たとえば，de@foo.barにメールを送るのであったら，

```
open(HANDLE,"|sendmail de¥@foo.bar");
while(メールが終わるまで){
    print HANDLE $コンテンツ;
}
close(HANDLE);
```

とするわけです。

ここで気をつけてほしいのは，sendmailのメールの送り先をなんでも書けるようにしてしまうと，それがセキュリティホールになりかねない，ということです。たとえば，

```
open(HANDLE,"| sendmail | 他プログラム");
```

としてしまうとほかのプログラムをサーバで起動できてしまいます。こんなことがないようにこの手のプログラムではメールの送り先をチェックして，正しいメールアドレスである場合だけ，メールを送るようにしておくべきです。幸い，今回のプログラムの場合，対象が文字電話をはじめとするPメールDXサービスのみですので，このサービスで必ず使われているメールアドレス，

「@pdx.ne.jp」

が含まれるものだけを受け付けるようにしました。また，メールのアドレスも，

```
sendmail "foo@hoge.co.jp"
```

となるように「"」で囲ってしまいます。もし，これで「|」のようなパイプライン文字を入れられても，与えられた文字列がひとつのメール転送先アドレスとみなされて，悪用されることはなくなるはずです。

ところで，このsendmailはとてもシンプルなメールソフトで，メールのヘッダなどは自動的には作成してくれませんし，文字種の変換なども行いません。つまり，メールのヘッダ部分はまず，sendmailをopenしたら，最初に自分で書き込まなくてはなりません。スクリプトwebmail.pl中では，

```
open (RPT,"|/usr/sbin/sendmail -Fwebmail ¥"$sendto¥"");
print RPT "To: $sendto¥n";
print RPT "From:
mojiden¥@multi.tsp.storage.cs.fujitsu.co.jp¥n";
print RPT "Subject: $url¥n";
print RPT "MIME-Type: 1.0¥n";
print RPT "X-Mailer: webmail 0.11¥n";
print RPT "Content-Type: text/plain; charset=ISO-2022-
JP¥n¥n";
```

としています。また，メールの内容はコンテンツヘッダ中でも宣言しているようにISO-2022-JP，つまりいわゆるJISコードに変換して書き込むことになります（メールはJISコードで書くこと，というのが一般常識ですし，それ以前の問題として，筆者が確認した限りではこのプログラムで正しく表示できるメールは文字電話端末ではJISコードかus-asciiで書かれたメールのみです）。プログラム中では，

```
print RPT Jcode->new($text)->h2z->jis;
```

というようにJcode.pmの機能を利用して，JISコードに変換して送信しています。

メール着信でプログラムを起動させる

さて，ここまでで，webmail.pl本体の説明は終わりました。このwebmail.plさえあれば，まだメールの着信で起動することはできませんが，メールで与えられるのと同等のデータを標準入力から入力してやると，文字電話に特定のURLのWebページをメールとして送ってみることができます。もし，お使いのWebサーバでtelnetなどが使えるのであれば，webmail.plを実行して，試してみてください（初めてこのスクリプトを実行する際には，

```
>chmod 755 webmail.pl
```

などとして，プログラムに実行権を与えておくことを忘れないように（私もしょっちゅう忘れますので，人のことはいえませんけど……）。

```
>webmail.pl
```

として実行します。すると，入力待ちになりますので，ここでメールに書かれている内容をキーボードで入れていきます。題名がURL，メールの送信元が送り先ですから，

```
Subject: http://www.softbank.co.jp/
From: foo@pdx.ne.jp
```

そして，１行改行して，「CTRL」+「D」を入力します。まもなく，文字電話のメール着信通知をONにしている場合は，端末のベルが鳴り，着信が知らされるはずです。メールを送受信してみると，ソフトバンクのトップページの内容を（かなりムダな改行がたくさんあってスカスカですが）見ることができるはずです。

あと残りの作業は，これが文字電話からのメール着信で，このスクリプトが起動し，メールを自動で返信させるようにするだけですね。

これには，今回のプログラム，webmail.plではsendmailの機能を使っています。

UNIXのシステムだと，SMTPサーバにsendmail，あるいはqmailを入

れていることが多いかと思いますが，これらにはメールを受信するとメールを自動で転送する機能を持っています。ホームディレクトリにsendmailを使用している場合なら「.forward」，qmailを使っている場合なら「.qmail」というファイルに転送用の設定を書くだけで，この機能を使うことができます。このメールの転送という言葉には「指定したプログラムへのメールの転送」という意味合いも含まれていて，たとえば，.forwardの場合，

foo@hoge.co.jp

と書いておくとそこにメールが転送されますが，ここにメールのアドレスではなく，先頭に「｜」という文字を書き，プログラムのパスを，

| /home/webm/webmail.pl

というように書いておくと自動的にプログラムを起動してくれるのです。

起動されたプログラムから見ると，到着したメールは標準入力から1行ずつ入力されているように見えます。つまり，

```
while(<>){
    1行ずつメールの処理;
}
```

というスクリプトを書いておくとメールの処理を行うことができるわけです。webmail.plでは，ここでメールをまとめて1個の変数に入れておいて，それからいろいろな処理をしています。

なお，.forwardファイルの書き方などについてですが，sendmailがインストールされているUNIXシステムなら，

man forward

で見ることができるはずです。ところで，幸いなことに，この手のヘルプファイルはたとえば，いくつかのPC用UNIXのヘルプファイルや翻訳ドキュメントが掲載されているWebサイトがありますので，この辺を見るようにすると参考になります。たとえば，PC用のUNIXとして有名どころのFreeBSDのサイト，

FreeBSD Project(Japan)
http://www.jp.freebsd.org/

の日本語マニュアル検索で「forword」を調べてみると以下のように載っています。

http://www.jp.freebsd.org/cgi/mroff.cgi?subdir=man&lc=1&cmd=&man=forward&dir=jpman-3.4.0%2Fman&sect=0
(※これはFreeBSD3.4の場合の「.forward」のマニュアルです。)

FORWARD(5)　FreeBSD File Formats Manual　　FORWARD(5)

名称
　forward - メールの転送指示

解説

.forward ファイルは，ユーザーのメールの転送先のメールアドレスもしくはプログラムのリストを格納します。このファイルが存在しない場合には，メール転送は行われません。行頭に標準のシェルパイプシンボル（|）をつけることにより，メールはプログラムの標準入力として転送されます。引数をコマンドに渡す場合には，行全体をクォートで括る必要があります。セキュリティ上の理由で，.forward ファイルはメール送付先のユーザーもしくは root が所有者である必要があり，ユーザーのシェルが /etc/shells にリストされている必要があります。

たとえば，.forward ファイルが以下の行を含む場合，

nobody@FreeBSD.org
"|/usr/bin/vacation nobody"

メールは nobody@FreeBSD.org に転送され，単一引数 nobody を伴ってプログラム /usr/bin/vacation に転送されます。

サンプルプログラムの使い方

以上で，「webmail.pl」と設定ファイル「.forward」の説明が完成したわけですね。

さて，付録CD-ROM中では「webmail.tgz」という名前で収録されているはずです。このファイルを解凍すると，webmailディレクトリ中に，

webmail.pl
forward

というファイルが展開されます。もし，UNIXマシン上で展開した場合には，実行権は付加されているはずです（「ls -l」として確認してみてください)。

これらのサンプルファイルには筆者の使った環境にパス名などが決め打ちになっているので，これらを編集してください。

まず，forwardには，

| /home/webmail/webmail.pl

と書かれていますので，あなたの使っているホームディレクトリを

> cd ~
> pwd

などとして確認してください。

/home/hoge

などと表示されているはずですので，そのパス名を使って，

| /home/hoge/webmail.pl

と書き換えます。

それから，webmail.plのほうですが，

#sendmailをオープン *** お使いの環境に合わせてパスを変更してください。
open (RPT,"|/usr/sbin/sendmail -Fwebmail ¥"$sendto¥"");

と書かれた部分があります。この，

|/usr/sbin/sendmail

の部分をお使いのsendmailプログラムのフルパスに書き換えます。シェル上で，

> which sendmail

などとすると，

/usr/sbin/sendmail

と表示されますので，これをそのまま使用します。

書き換え終わったら，これらのファイルをホームディレクトリに転送します。

cp webmail ~
cp forward ~/.forward

のようにすればOKです。

そうそう，今回使用した各種モジュールを事前にインストールするのをお忘れなく。もし，CPANシェルが使える環境にあるならば，

$ perl -MCPAN -e shell

として，

cpan> install Jcode
cpan> install Jcode
cpan> install LWP::UserAgent;
cpan> install HTTP::Request;
cpan> install HTTP::Response;

とすればモジュールを簡単にインストールできるでしょう。もし，サーバ管理者でないのならば，使っているPerlがPerl5であることを確認してから，上記の操作を管理者にしてもらうのが簡単でいいでしょうね。

というわけで，いかがでしたでしょうか？

なお，テスト用として，このプログラム「webmail」を筆者のホームページに置いておきました。もし，文字電話，あるいはH"などのPメールDX対応PHSをお持ちの場合は，試してみてください（本誌発売から1カ月くらいは設置してあるんじゃないかと思いますが，サービスが停止などしてしまった場合はご容赦ください)。

webmail@shippo.co.jp

です。PメールDXのセンターメールで，題名に見たいURLを英数半角文字で書いて送ると，しばらくして，メールでそのURLの内容がメールとして送られてくるはずです。

オルタナピコピコC言語講座　#1（第2回）
スネークゲームによるC言語の実践的学習

飯田 伸一　Iida Shinichi

簡単なゲームsの作成を通してC言語を学んでいきましょう。今回は前回制作したスネークゲームを対戦ゲームに仕上げていきます。同時にポインタや関数の基礎も身に着けてください。

「ああ，昔はよかったなあ」

……という，なんだかノスタルジーめいた会話を，比較的若いと思われる層から耳にする機会が多くなったことに驚いている。過去に起こった失敗などのイヤな出来事を記憶から浄化して，美化するのがノスタルジーの魔法である（これは一概に悪いこととはいえないが）。このような枯れた言葉を発する輩は，現在の自分の置かれている状況と比べて，まだ過去の状況のほうがマシであったと勘違いしてしまっているのだろう。

似たような状況でよく聞くものに「（真顔で）あの先生は，厳しかったけれどよい先生だった」というのがある※1。これは当時，明らかにイヤでしかたがなかったこと（体罰とかね）を忘れて，「よい先生だった」ということだけが美化されて残った結果「厳しい先生≒熱心でよい先生」ということになってしまったのではないかと思うのだ（この毒花な公式には，もちろん例外があるのだが）。

このようなしょうもないノスタルジーばかりにとらわれていると，思考停止に陥ってしまい，どうも人間としての成長がこれ以上望めないような気がする。

この講座も「ピコピコ」という名前からして懐古主義的ではないか，と思われるかもしれない。僕がピコピコゲームを題材にするのは，C言語を学ぶのに優れた手段のひとつであり，それはいまでも十分に有効だと考えているからだ。そこのところが冒頭で述べた気色悪いノスタルジーとの大きな違いだ。問題は，現場感覚というか，時代性にあっているかどうかなのである。Oh!Xも単なるノスタルジーではなく，本当に「私たち」に必要とされて復刊を果たした。その思想が時代を超越し，いまでも有効であることに驚く。

※1　ちなみに僕の中学生時代の数学教師は，軍隊じこみ（自称・特攻隊の生き残り）のそれはそれは，素晴らしい前近代的な教育理念をお持ちであった。なんたって数学の計算を間違えるごとに，竹刀 or 拳（その日の機嫌によりウェポンの装備が変わる）で頭をブン殴るのだから……。いままでの人生で，最大のアドレナリン放出量を誇った時期だ。

スネークゲーム"Hello World!"完結編

前回は，単純なヘビゲーム"Snaky1"を例として挙げ，そのソースリストを読み進めながら，C言語とはどういうものなのか，またゲームを作るにはどうしたらいいのかという基礎的なところを解説した。僕としては，手軽に学べるC言語のチュートリアル的なものを目指して書いたつもりだ（導入という意味では，やや冗長ではあったかもしれないが）。

今回はその続きとして，前回の最後に予告したことを中心に，まずはSnakyの2プレイヤーバージョン（以下，1P，2P）を例に挙げて各事項を解説し，後半部分では，関数について重要な部分を，まとめと補完の意味を含めて書く。

なお，前回と同様に今回のサンプルの制作も引き続き，Botchy氏が作成したフリーライブラリ"el"を使わせていただいた（多謝）。このライブラリを使用すると，複雑なDirectXの機能を比較的簡単に利用できる。対応しているコンパイラは，マイクロソフトのVisual C++（ver4.0以上），インプライズのBorland C++，C++Builder（ソース先頭のコメントを外し#define BCB WinMainを有効にすること）である。このelついての詳細は，elヘルプを参照してほしい。

※elとelヘルプはCD-ROMに収録。

サンプルゲームの遊び方

"Snaky2"（**画面1**）は，2匹のヘビの体がどんどん伸びていくので，相手の行く手をそれぞれWASDキー（1P）とカーソルキー（2P）で阻んで袋小路に叩き込むという，昔どこかのゲームセンターで見たことがあるようなスネークゲームだ。ちなみに僕は，学校をサボってこのゲームをしているときに，補導されて（以下略），……なにか痛くてイヤなことを思い出してしまった※2。ので，とっとと今回のサンプルプログラム（**リスト1**）の解説に移ることにしよう。前回と同様，本文とソースリストをリンクさせて説明していくことにする。

なお，前回を読んでいない方でも，理解できるように末筆にリファレンス（のようなもの）を書いておいたので，必要に応じてそちらを読んでいただきたい。

※2　本文とは全然関係ないが，痛いといえば，どうしてヘビメタ系のギタリストは，ギターソロのときに「痛そうな顔」をするのだろうか？　謎だ……（"Burrrnn!!!"my fingers!ってか？）。

プログラムについて

基本的なプログラムの流れは，前回のサンプルゲーム"Snaky1"とあまり変わりはない。だが実際にリストを読んでみると，2P版を実現するために，プログラムの変更をかなりの部分にわたって施していることに気がついてもらえるだろう。この変更点を中心に，各事項の解説をしていこう。

※"Snaky"をコンパイルする場合は，2つのWaveファイルを作業フォルダへインポートすること。



画面1

リスト1

```
/******************************************************************************/
/*                                                                            */
/*  "SNAKY2"...List1  '99 9/20 Programed by.飯田伸一                          */
/*                                                                            */
/******************************************************************************/


//  以下のコメントをはずすと," Borland C++ Builder3" でコンパイルが可能になる
//#define BCB WinMain

#include "el.h"

//  画面No.を定義
#define MAIN_SCREEN 1
#define OVER_SCREEN 2

//  上の画面Noと対になる，画面用の関数のプロトタイプ宣言
void MainScreen(void);
void OverScreen(void);

//  ゲーム処理中の初期化関数のプロトタイプ宣言
void InitFunc(void);

//  サウンドオブジェクトの定義
DSOBJ PicoWAV;   //  ピコピコ音
DSOBJ OverWAV;   //  ミスした時の効果音

//  ゲーム初期化関数のプロトタイプ宣言
void InitFunc(void);

//  ゲーム画面の最大値
const int MAX_X=38;
const int MAX_Y=28;

// プレイヤー情報
struct player{
   int D,Px,Py;            //  描画座標
   int Up,Down,Left,Right;   //  キー入力状態保持
   float WaitCounter;
}Player[2];

enum dir{UP,DOWN,LEFT,RIGHT};

//  キー操作情報
const int Key[2][4]={
  'W','S','A','D',                //Player1
  VK_UP,VK_DOWN,VK_LEFT,VK_RIGHT //Player2
};

int i,j;
char ImaginaryScreen[MAX_X][MAX_Y];

/******************************************************************************/
/*                                                                            */
/*  メイン関数                                                                */
/*                                                                            */
/******************************************************************************/


int elMain("Snaky2")
{
   elWindow(640,480,FALSE);
   elSetWindow(RUN_NOTOP|RUN_INACTIVE);
   elLoop()
   {
       elSetScreen(MAIN_SCREEN,MainScreen());
       elSetScreen(OVER_SCREEN,OverScreen());
   }

   elExitMain();
}

/******************************************************************************/
/*                                                                            */
/*  ウィンドウ生成関数                                                        */
/*                                                                            */
/******************************************************************************/


void elCreate(void)
{
   elDraw::Screen(640,480);
   //  サウンドオブジェクトの読みこみ
   PicoWAV=elSound::LoadObject("BGM.WAV");
   OverWAV=elSound::LoadObject("OVER.WAV");
   elCallScreen(MAIN_SCREEN);
}

/******************************************************************************/
/*                                                                            */
/*  キーボード関数                                                            */
/*                                                                            */
/******************************************************************************/


void elKeyboard(void)
{
   case VK_ESCAPE:
   elDraw::Exit();
   break;
   elExitKeyboard();
}

/******************************************************************************/
/*                                                                            */
/*  イベント関数                                                              */
/*                                                                            */
/******************************************************************************/


long elEvent(void)
{
   elExitEvent();
}

/******************************************************************************/
/*                                                                            */
/*  メイン画面関数                                                            */
/*                                                                            */
/******************************************************************************/


void MainScreen(void)
{
   //  画面が切り替わった直後の処理
   if (elChangeScreen())
   {
       //  初期化処理関数の呼び出す
       InitFunc();

       //  ピコピコ効果音を再生（ループ）
       elSound::Loop(PicoWAV);
   }

   //  仮想画面を表示する
   elFont::Begin(GOTHIC,24);
   elFont::Color(RGB(255,255,255));

   for(i=0;i<MAX_Y;i++){
       for(j=0;j<MAX_X;j++){
           sprintf(Buffer,"%c",ImaginaryScreen[j][i]);
           elFont::Draw(j*17,i*17,Buffer);
       }
   }

   elFont::Before();

   //  ループで2P分の処理をする
   for(i=0;i<2;i++){

       //  プレイヤー最新座標にヘビキャラクタを代入
       ImaginaryScreen[Player[i].Px][Player[i].Py]='$'+i*('\\'-'$');

       //  キーの入力状態を取得
       elSystem::GetKey(Key[i][0],&Player[i].Up);
       elSystem::GetKey(Key[i][1],&Player[i].Down);
       elSystem::GetKey(Key[i][2],&Player[i].Left);
       elSystem::GetKey(Key[i][3],&Player[i].Right);

       //  押されたキーによりキャラクタの方向を決定
       if (Player[i].Up==PUSH_KEY && Player[i].D!=DOWN)
           Player[i].D=UP;     //「↑」
       if (Player[i].Down==PUSH_KEY && Player[i].D!=UP)
           Player[i].D=DOWN;   //「↓」
       if (Player[i].Left==PUSH_KEY && Player[i].D!=RIGHT)
           Player[i].D=LEFT;   //「←」
       if (Player[i].Right==PUSH_KEY && Player[i].D!=LEFT)
           Player[i].D=RIGHT;  //「→」

       //  一定時間が経過したら処理をする（ウェイト部分）
       if (Player[i].WaitCounter>=F(15)){

           //  方向によって座標を増減
           Player[i].Px+=(Player[i].D==RIGHT)-(Player[i].D==LEFT);
           Player[i].Py+=(Player[i].D==DOWN)-(Player[i].D==UP);

           //  ゲームオーバーか？
           if (ImaginaryScreen[Player[i].Px][Player[i].Py]=='$'
               || ImaginaryScreen[Player[i].Px][Player[i].Py]=='\\'
               || ImaginaryScreen[Player[i].Px][Player[i].Py]=='#'){

               //  ゲームオーバ画面処理関数へ
               elCallScreen(OVER_SCREEN);
               break;
           }

           //  ウェイトカウンタを再初期化する
           Player[i].WaitCounter=F(0);
       }

       // ウェイトカウンタを加算する
       Player[i].WaitCounter+=FrameTime*F(100);
   }
```

```
    // 画面の更新
    elDraw::Refresh();
}

/*****************************************************************************/
/*                                                                           */
/*  ゲームオーバ画面関数                                                       */
/*                                                                           */
/*****************************************************************************/

void OverScreen(void)
{
    if (elChangeScreen())
    {
        // BGMの演奏を止める
        elSound::Stop(PicoWAV);

        // ゲームオーバ時の効果音
        elSound::Play(OverWAV);
    }

    elFont::Begin(GOTHIC,24);
    elFont::Color(RGB(255,0,0));
    elFont::Draw(Player[i].Px*17,Player[i].Py*17,"X");
    elFont::Draw(234,220,"Hit Space key!");
    elFont::Before();
    elDraw::Refresh();

    // 再びメイン画面処理関数へ
    if (PushKey==VK_SPACE){
        elCallScreen(MAIN_SCREEN);
    }
}
```

```
// ゲーム初期化関数
void InitFunc(void)
{
    // 画面をクリア
    elDraw::Clear();

    // プレイヤー情報の初期化
    for(i=0;i<2;i++){
        Player[i].Px=3+(i*29);
        Player[i].Py=3+(i*19);
        Player[i].D=1+i;
        Player[i].WaitCounter=F(0);

        Player[i].Up=FREE_KEY;
        Player[i].Down=FREE_KEY;
        Player[i].Left=FREE_KEY;
        Player[i].Right=FREE_KEY;
    }

    // 仮想画面の初期化
    for(i=0;i<MAX_Y;i++){
        for(j=0;j<MAX_X;j++){
            ImaginaryScreen[j][i]='#';
        }
    }

    for(i=1;i<MAX_Y-1;i++){
        for(j=1;j<MAX_X-1;j++){
            ImaginaryScreen[j][i]=' ';
        }
    }
}
```

関数について

前回は，いろいろな関数の組み合わせでプログラムが構成されているということと，計算式以外で () が含まれるものは，すべて関数であるということを解説した。関数を作成し，使用するためには通常，

プロトタイプ宣言
関数定義
関数の呼び出し

という 3 つのステップが必要となる。では，ソース中のゲームの初期化を行う関数InitFuncを例に説明してみよう。

```
void InitFunc(void);
```

まずは，プロトタイプ宣言※3。これはコンパイラに前もって「こういう関数を使いますよ」と伝えている部分だ。"void"の部分は関数の値の受け渡しに使用する部分だが，詳しい解説は後述に譲るとして，ここではまだ意識しないことにする。

```
void InitFunc(void)
{
  関数本体
}
```

Windows上でのプログラムの流れ

COLUMN 1

従来のMS-DOSのCで作成したプログラムは，最初にmain関数から起動され，基本的にはmain関数内に記述した処理を逐次実行していく。一方，Windows上でのプログラムは，WinMain関数から起動して，RUN関数というところでループ(繰り返し)をしていて，キーボードやマウス操作などの動作が起きると，それにあわせた関数が実行される。

つまり，前者はプログラマが書いたとおりにプログラムが走る「手続き型プログラム」なのに対し，後者は，マウスなどが押された場合，「マウスが押された」というメッセージがWindowsに伝えられ，それに対応したメッセージ関数が実行される「メッセージ駆動型プログラム」(メッセージドリブンプログラム) というものだ (このへんはだいたいわかることを目標に，簡単にまとめているので，重箱の隅をつつくようなことはしないでね)。

ソースリストの最後のほうにあるのが，InitFunc関数の関数定義部分で，"{"と"}"で囲まれたブロックの中に，その関数の処理内容を記述する。

```
InitFunc();
```

これで，その関数の機能が実行されることになる。このようにしてC言語では，自分で関数を作ったり，既製の関数 (標準関数など) を利用したりして，プログラムを組んでいくのだ。

ソース中にあるelMain関数は，一般のC言語でいうところのmain関数 (WindowsではWinMain関数) にあたり，プログラムが起動されると最初に動作する関数だ。この関数は呼び出して使用するのではなく，WinMain関数の代わりとして使用する (Windows上のプログラムの動きについては，コラム1を参照してほしい)。

※3　C++になってから，プロトタイプ宣言は必須となった。このメリットは，コンパイラに関数の戻り値の引数の数や型を伝えることによって，型の違う値を関数に渡そうとすると起きる「型の不一致による不具合」をコンパイル時に発見できることである (ANSI C以前の古いCの規格では，関数のプロトタイプ宣言がないものがあり，その場合は型のチェックをすることができなかった)。

キーコードが保存されている 2 次元配列"Key"

```
const int Key[2][4]={
'W','S','A','D',
VK_UP,VK_DOWN,VK_LEFT,VK_RIGHT
};
```

ここでは，2 次元配列を用いて，ヘビを上下左右に操作するときに使う 4 つのキーコードをconst,int型で宣言・定義している。つまり，添字 [0][0] から [0][3] まで順番に"W,S,A,D" (1P) が，[1][0] から [1][3] までは，カーソルキー (2P) の仮想キーコードが入ることになる。

このように配列を使用することによって，2P分のキー入力を簡潔に記述することができる。こうした配列や，後述する構造体を利用して，それぞれの処理を 2 回分のループさせることによって，2P版を実現させている。

ちなみに，最初の配列の要素は，[ ] というように 省略することができる。これは，あとで初期化データを追加することになっても，配列の要素数を書き直す必要がなくなるというメリットがある。

構造体・struct

プレイヤーの情報（スコア，X,Y座標……）などのいくつかの変数が同じものを示す場合は，まとめてひとつのグループにまとめておくとわかりやすくて，変数を扱う際にとても便利だ。この機能を実現するのが構造体と呼ばれているものだ。
構造体を使用するためには，構造体の定義と宣言を行う。

例：
```
struct character{
 int x;
 int y;
  ……
}Chr ;
```

struct（structue：構造という意味）は，構造体を使うための型だ。次のchracterと書かれた部分は構造体タグというもので，ここで定義された構造体を識別するための名前である。あとは，使いたい変数（メンバと呼ぶ）を{ }内に宣言させてやればいい。最後のChr;でその構造体の変数を宣言する。
あとは普通の変数と同じように扱える。たとえば，この構造体のメンバxに1を代入したい場合は，"Chr.x=1;"（構造体変数名.メンバ名）のように記述する。
ここでは構造体の定義と宣言を一緒にしたが，変数名"Chr"を定義部から削り，

```
struct character Chr;（struct 構造体タグ 変数名;）
```

のように定義と宣言を分ける方法もある。
では実際にプログラムではどういう風に使われているのかを見ていこう。

```
struct player{
int D;
int Px,Py;
……
} Player[2];
```

最後の"Player [2];"の部分で，この構造体を宣言しているのだが，配列を用いることによって，それぞれのメンバが2つずつ値を確保し，結局2P分の構造体を持つことになる（構造体の配列）。たとえば，1PのX座標を扱いたいときは"Player [0].Px"というように書けばいい。
せっかく構造体について説明をしたので，ここでクラスのことも少し触れよう。構造体とクラスはとても似ている※4。構造体は，変数にメンバを持っていたが，クラスには，変数に加え関数もメンバ（メソッドと呼ぶ）に持てる。この講座ではelの公開クラスを使用するだけなので，あまりクラスについて意識する場面は少ない。そこで，クラスについての詳細は，C++オンラインヘルプや市販の解説書を参照してほしい。

※4　C++は，C言語の上位互換言語だ。基本的には「C＋クラス」の機能を持ったものだが，そのほかにもさまざまな新しい機能が追加されている。この講座では，主にC言語を扱っているが，C++の機能を使用したほうが便利な場合には，そちらを採用している。

列挙型・enum

構造体と同じような記述をするものに列挙型（enumeration）というものがある。これは，複数のint型の値を順番に並んだ形で同時に定義して，初期化をするものだ。

```
enum dir{UP,DOWN,LEFT,RIGHT};
```

enum型は列挙型を使うための型で，次に列挙型タグ，そして列挙したい単語（列挙定数）を書く。この例では，"UP=0,DOWN=1,LEFT=2,RIGHT=3"という具合に，0から始まり，1ずつ大きくなる値が順番に割り当てられていく。0からではなく，2から値を始めたい場合は，

```
{UP=2,DOWN,LEFT,RIGHT};
```

とすればDOWNには3が入って，以降4，5と数値がそれぞれに割り当てられる。
このように列挙型を使用するのは，ヘビの移動方向を"0－3"で表すよりも，"UP,DOWN,LEFT,RIGHT"（上下左右）という単語で表したほうが，プログラムが読みやすくなるからだ。単にひとつの単語を数値に置き換えるのならば，前回説明した#defineやconstを使う方法があるが，このケースのように，関連のある連続的な単語をまとめて定義したい場合は，列挙型を使用したほうがなにかと便利だ。

メイン画面関数内の解説

```
ImaginaryScreen[Player[i].Px][Player[i].Py]='$'+i＊('¥¥'-'$');
```

ここからいよいよ，メイン画面関数のループ内の解説に移る。まずは，仮想画面に各プレイヤーの最新座標（Player構造体のメンバPx，Py）にヘビキャラクタを代入している部分だ。初回のループでは，ループカウンタiの値が0なので，()内は計算されず，1Pのキャラクタ「$」が代入され，2回目のループでは，iの値が1となり，()内の演算が反映され，2Pのキャラクタ「¥」が代入されることになる。
ソースにもあるとおりに，制御文字「¥」を代入するためには，単純に"¥¥"をひとつ足せばいい。これは，「"」や「'」などを扱いたい場合も同様である。

キー入力について

```
elSystem::GetKey(Key[i][0],&Player[i].Up);
elSystem::GetKey(Key[i][1],&Player[i].Down);
elSystem::GetKey(Key[i][2],&Player[i].Left);
elSystem::GetKey(Key[i][3],&Player[i].Right);
```

キー入力は，elのelSystemクラス中のGetKey関数で行っている。判定したいキーを前述の配列"Key"から参照して，あらかじめ用意しておいた変数（構造体Playerのメンバ"Up,Down,Left,Right"）にキー状態を保存している。

```
if(Player[i].Up==PUSH_KEY && Player[i].D!=DOWN)
  Player[i].D=UP;
……
```

ここのif文でキー入力状態に応じて（PUSH_KEYはキーが押されている状態），ヘビの進行方向を決定している。あと，前回では反対方向にキーを押すだけでゲームが終わってしまうという，少しマヌケな部分があったが，論理演算子※5のAND（&&）を用いて，進行方向から逆のキーが押された場合は，そのキー入力を無視するように修正した。

※5　このように，プログラム中のif文などで，複数の条件文の間で使われるのが論理演算子だ。"OR"（||）で，式のひとつでも正しければ真になり，"AND"（&&）で式の両方，"NOT"（!）でどちらも正しくないときに真となる。

ifを使わない方法

前回では，ヘビのXY座標の増減にはif文を使ったり，switch文で置き換えてみたりして処理をしていた。ここでは，もうひとつの選択肢として，

ifを使わないで，値の増減をする方法を紹介する。

```
Player[i].Px+=(Player[i].D==RIGHT)-(Player[i].D==LEFT);
```

左右のカッコの中で，比較演算をしている。まず左の()の中で，ヘビの方向が保存されているメンバDを比較して，正しければ1（真）が演算結果として残って"Player [i] .Px+=1;"となり，X座標の値がひとつ増える。逆に右の()の比較が正しければ，"Player [i] .Px+=-1;"となって，X座標の値が引かれることになる。

このように()だけでも比較演算ができるのだが，今回のように単純な場合を除いて，複雑なものになると，かなりわかりづらくなってしまうので，あまり多用しないほうがいいだろう。

機種に依存しない処理の仕方

コンピュータには，CPUなどの違いにより，処理速度にはそれぞれバラつきがある。ゲームにおいては，どのマシンでも均一な処理速度になるように，プログラムが進行していくことが望ましい。前回のサンプルゲームでは，処理速度を一切考慮していなかったので，もしかしたらこの本が出るころには，速すぎてゲームにならないようなCPUが出荷されているかもしれない（ちなみに僕のマシンはCerlon400MHzを少しオーバクロックさせて約450MHzで動作させている）。

昔は，開発するマシンがある程度明確だったので，そのマシンにあわせてウェイトをかけたり，速いマシンだとより綺麗な動作をさせたりして，それぞれゲームをするのに支障のない速度で動いていたのだが[※6]，いまは動作ターゲットとなるマシンの速度はバラバラで，ウェイトをかけて調節するのは，ちょっと大変になってきた。

幸いにもelでは，どのマシンにおいても，比較的簡単にほぼ均一な処理速度を得るための公開変数が用意されているので，それを使ってみよう。

elでは，フレーム（1回の描画）にかかった時間を，FrameTime変数に保存している。それを利用すれば，速度に依存しないプログラムが作れる。

たとえば，1秒間に100ドット進むキャラクタがいた場合，以下のように記述する。変数Pxには，キャラクタの座標が入る。floatにしたのは，精度を上げるためだ（FrameTime変数はfloat型）。

例：
```
float Px=0.0f;
Px+=FrameTime * 100.0f;
```

Snaky2では，キャラクタごとではなく，ゲーム全体の速度を一定にしたいので，一定時間が経過したあとに，それぞれのキャラクタ座標の増減やゲームオーバチェックの処理をするようにしている。

※6　逆に，ゲームにならないほど遅いトンデモなゲームもあった。某ステイックのレ○ク○とか……（以下自粛）。

効果音・Wave

効果音を出したい場合は，主にWaveというフォーマットを使用する[※7]。elでは，DirectSound（Direct-X）の機能も簡単に扱えるようにelSoundというクラスが用意されているので，これを使ってWaveの音を再生してみよう。

```
DSOBJ PicoWAV;
DSOBJ OverWAV;
```

通常，効果音はWaveファイルに保存されているが，このままではなにかと扱いづらいので，任意の変数に入れることにする。DSOBJ（Direct Sound Objectの略）とは，Waveデータが保存されるオブジェクト（構造体）型を示し，PicoWAVでBGMのピコピコ音，OverWAVでゲームオーバ時の効果音が入ることになる。

```
PicoWAV=elSound::LoadObject("BGM.WAV");
OverWAV=elSound::LoadObject("OVER.WAV");
```

これで，用意された変数にWaveデータがロードされる。これは単純に「DSOBJ型の変数=elSound::LoadObject（"効果音が保存されているファイル名.WAV"）;」と記述すればいい。

```
elSound::Play(OverWAV);
```

あとは，elSoundのPlay関数で，任意のWaveデータを指定してやるだけだ。

```
elSound::Loop(PicoWAV);
```

BGMのように効果音を何度も繰り返し再生したい場合はPlay関数の代わりにLoop関数を使用する。ループしている効果音を止めたいときは，Stop関数を使う。

ちなみに，今回は，「ピコピコ」という非常に短い音なので，WaveをBGMとして流しているが，本格的にBGMを扱いたい場合は，MIDIというフォーマットを使用するのが一般的だ（これはかなり乱暴なたとえだが，X68000ユーザにとっては「Wave=AD PCM」，「MIDI≒FM音源」といったほうがわかりやすいだろうか）。このMIDIというものは，次回のサンプルゲームで実装する予定なので，そのときに改めて解説するつもりだ。

※7　Wave（効果音）を作るためにはなんらかのツールが必要だ。Windowsに付属のサウンドレコーダか，サウンドボードに付属しているものを利用しよう。

関数についての補足事項

ここからは，前書きの部分で予告したとおりに，関数を実践的に学んでもらうために，新たにサンプルプログラムを挙げて，関数について重要と思える部分を補完の意味を含めて，一気にまとめて解説する。特にポインタについて説明している後半の箇所では，やや難解に思える部分があるかもしれない。その場合は，一度に全部を覚えようとはしないで，わかるところだけを少しずつ覚えていってほしい[※8]。

※8　ぶっちゃけた話をすると，その時点では必要のない細かい部分をあまり気にしないのが，プログラミング上達の秘訣ともいえる。すなわち，本筋に関係のない枝葉ばかりにこだわっていると，余計なこと気をとられて，いつまでたってもプログラムが完成しないという本末転倒な事態に陥りがちになってしまうのである（ただでさえいまは，覚えなければならないことが多すぎるからねぇ……）。

まずは，関数の定義の書式を図1にまとめてみる。

では，前述したとおりに，ここで関数のサンプルプログラム（リスト2）の



画面2

リスト2

```
//関数サンプル (List2)

#include "el.h"

#define MAIN_SCREEN 1

void MainScreen(void);

int Func1(int,int,int);
void Func2(int*,int*,int*);

int elMain("関数サンプル")
{
    elWindow(320,240,FALSE);
    elSetWindow(RUN_NOTOP|RUN_INACTIVE);
    elLoop()
    {
        elSetScreen(MAIN_SCREEN,MainScreen());
    }
    elExitMain();
}

void elCreate(void)
{
    elDraw::Screen(320,240);
    elCallScreen(MAIN_SCREEN);
}

void elKeyboard(void)
{
    case VK_ESCAPE:
    elDraw::Exit();
    break;
    elExitKeyboard();
}

long elEvent(void)
{
    elExitEvent();
}

void MainScreen(void)
{
    elDraw::Clear();

    int a=10,b=10,c=0;

    SHOW2(0,0,"a*b=%d",Func1(a,b,c));
    SHOW2(0,11,"a=%d b=%d c=%d",a,b,c);

    Func2(&a,&b,&c);
    SHOW2(0,21,"a=%d b=%d c=%d",a,b,c);

    elDraw::Refresh();
}

int Func1(int x,int y,int z)
{
    z=x*y;
    return (z);
}

void Func2(int *x,int *y,int *z)
{
    *x=20;
    *y=20;
    *z=20;
}
```

説明をしよう。実行画面（画面2）を眺めながら考えてほしい。まずは，関数Func1()の定義部分を見ていこう。

```
int Func1(int x,int y,int z)
{
z=x * y;
return (z);
}
```

最初に3つのint型の引数x，y，zを取り，次にxとyを乗算したあと，その結果をzに代入している。そして，最後のreturn文でその演算結果を戻り値として，呼び出し元の関数へと返却している。この関数定義部分は，図1を参照しながら，読み進めると理解しやすいだろう。




```
int a=10,b=10,c=0;
SHOW2(0,0,"a * b=%d",Func1(a,b,c));
```

MainScreen関数の中で，Func1関数の呼び出しを行っている。SHOW2とは，文字表示をするel公開関数で※9，ここではFunc1の処理結果（戻り値）である100を画面に表示している。

```
SHOW2(0,11,"a=%d b=%d c=%d",a,b,c);
```

この実行結果を見てもらえば，引数で渡した変数cの値が0のままで変化がないのに気がつくだろう。

関数定義部分での引数は（ここではx，y，zのこと），仮引数と呼ばれているもので，関数が呼ばれるごとに一時的な退避場所（スタック）へコピー

図1

```
①戻り値の型 ②関数名 ③(引数1，引数2……)
{
        ④処理
        ⑤return文
{
```

① 関数が値を返すときの型を指定。②関数の識別子※10。③関数が受け取る値（引数）の型と，変数名。ちなみに，この引数の変数名は呼び元と同じである必要はない。④関数が行う処理内容を記述。⑤return文で，関数の処理を終えたときに返す値を指定。

※なお，"void"という型を指定すると，それぞれ戻り値，引数を持たない関数ということになる。

され，そのコピーされた値を関数内では使用する。つまり，スタックに渡された仮引数は，その関数の処理が終了すると消滅してしまうので，なんら呼び出し元にはその結果が反映されない。それで，return文で戻り値（処理結果）を返しているというわけだ。

しかしこの方法では，ひとつの戻り値しか返すことができないという制限がある。それで，引数のアドレスを引数に渡してやれば，その関数内部だけではなく，呼び出し元にも処理の結果を反映させることができる。そのアドレス渡しを理解するためには，ポインタの知識が必要となるので，簡単に解説することにする。

※9　関数には，任意の名前をつけることができる。ただし，あらかじめ定義されている関数（予約語）や，先頭の文字は数字にしてはいけないなどの決まりがある。これは，どの関数，変数，構造体の識別子にも共通していえることだ。
※10　ちなみにCのprintfとほぼ同じ機能を持っている。

ポインタの考え方

いまはどうだかはわからないが，昔，ポインタはC言語習得の第一の壁であると呼ばれていた。確かに，細かいところまでを理解（解説も）しようとすると難しい部分があるのは事実だ。しかし，要点さえつかんでしまえば，あとは驚くほど簡単に理解できる。その理解するための「コツ」みたいなものを書きたいと思う。

コンピュータには，メモリ（RAM）が搭載されている。メモリとは0から順番に番号のつく連続した「箱」のようもので，さまざまなデータを入れることができる。箱の番号がアドレス，箱の中身がデータという感じだ。たとえば，"Hang the DJ!"という文字列データがメモリに入っていて，仮に0からアドレスが始まるとすると，図2のような感じになる（この例文に深い意味はないので，あしからず）。ポインタ変数には，データ（箱の中身）ではなく，アドレス（箱の番号）が任意の変数に代入されている。つまりポインタとは，データがどこにあるのかを指し示すものといえる。

表1　主な変数表

Key……2プレイヤー分のキーコード
ImaginaryScreen……仮想画面
PicoWAV……ピコピコ効果音(BGM用)
OverWAV……ゲームオーバー時の効果音
Player[n]構造体のメンバ
D,Px,Py……ヘビの向きと座標
Up,Down,Left,Right……キー入力状態を保持
WaitCounter……ゲームのウェイトカウンタ

ポインタ使用についての補足

COLUMN 2

前回説明した文字配列でポインタとの違いを比べてみよう。

```
char a[5]="Hi!";       char *b="Bye";
100 'H'                50  '100'←変数bの中身で"Bye"
                             の格納先アドレス
101 'I'                100  'B'
102 '!'                101  'y'
103 'Null'             102  'e'      103 'Null'
```

※仮に文字列の開始アドレスを100とした場合のメモリの内容。これは，見たとおりに変数宣言の際に＊（アスタリスク）をつけると変数をポインタとして宣言となる。おわかりだろうか，ポインタ変数はそのデータのアドレスが入っていて，間接的にデータをアクセスするのだ。

ポインタ変数を文字列と同様に参照したい場合は，「＊(b+1)」とすれば，'y'を読み出すことができる……。以上がポインタの基礎（のキソ？）の説明だ。ポインタについて詳しく書こうとするとそれだけで，誌面が尽きてしまうので，もしわからない部分があったら（多分あるとは思うが），Cの解説書をひもとかれたい。

さて，話は関数に戻って，実際にポインタを用いて関数の値の受け渡しをしてみよう。

```
void Func2(int *x,int *y,int *z)
{
*x=20;
*y=20;
*z=20;
}
```

変数宣言をするとき，その変数名の前に"＊"（アスタリスク）をつけるとポインタの宣言になる。上記の関数のようにポインタ宣言された変数は，以降"＊"（間接演算子）をつけることによって，その変数の値を処理することができる。なんだか意味が逆に思えるかもしれないが，ここで"＊"をつけないで，"x=20;"とするとx変数が入っているアドレスそのものを変更してしまうことになるので注意。

```
Func2(&a,&b,&c);
SHOW2(0,21,"a=%d b=%d c=%d",a,b,c);
```

変数の入っているアドレスの値を渡す場合は，"&"（アンパサンド）をつける。このようにポインタ変数によって，引数のアドレスがスタックへ渡ることになり，そのスタックの内容は，変数a,b,cの入っているアドレスを指しているので，Func2関数内の変数の変更も直接プログラム本体へと反映されることとなる。なんだかややこしくなってきたので，図3にまとめてみよう。

まとめてみると，主な関数へ引数の受け渡しには，

- 変数の値をそのまま渡す
- 変数のアドレスを渡す
- グローバル変数を使用する

の3つがある。

ちなみに，今回のサンプルゲームでは，プログラムサイズが小さいことと，共通して使用する変数が多いという理由でグローバル変数を利用して

図2　メモリの概念図

0	1	2	3	4	5	6	7	8	9	10	11	……アドレス(箱の号)
[H]	[a]	[n]	[g]	[ ]	[t]	[h]	[e]	[ ]	[D]	[J]	[!]	……データ(箱の中身)

図3

仮に変数が入っているアドレスが100から始まるとする。

	アドレス			スタック
Func1	100	10（aの値）	→	10（渡された変数の値をコピー）
	104	10（bの値）	→	10（同上）
	109	0（cの値）	→	0（同上）
Func2	100	20（aの値）	→	100（変数aのアドレス）
	104	20（bの値）	→	104（変数bのアドレス）
	109	20（cの値）	→	109（変数cのアドレス）

※Func1関数は，処理が終了するとスタックの内容が破棄されるので，変数a,b,cの内容には影響を与えない。

いる。逆をいうと，大きなプログラムを組む場合は，ここで説明した引数を関数に渡す方法を用いたほうが，あとあとプログラムを変更するときに，さほど混乱せずにすむので便利だ。中には「グローバル変数は使うべきではない」とまで書いてあるCの解説書があって，それはそれで一理あるのだが，前述したとおりにケースバイケースで臨機応変に使い分ければいいと思う。

```
int Func1(int,int,int);
void Func2(int *,int *,int *);
```

これは，リスト先頭にあるプロトタイプ宣言。()内のintは，呼び出し元から受け取るパラメータで引数の型を示し，先頭のintで，その関数の処理が終了した際の型を設定している。

Bye bye bad trip ooh bye bye...

前後編2回にわたって解説したSnakyシリーズであるが，前回の1P版では，Cの基礎的な事項を学んでもらい，そして今回の2P版で，構造体やそのほかの重要な部分をまとめて解説してみたのだが，どうだっただろうか。この連載によって，Cでゲームを作りたいが，どうしたらよいかわからない方に，ヒントというか，きっかけみたいなものを見つけてもらえれば幸いだ。あとは，サンプルゲームを改造するなり，自分でゲームを作ったりする過程で，必要に応じて細かいことを覚えていってほしい(完成したゲームは，ぜひOh!Xに投稿しよう)。

次回からは，ピコピコ実践編として，キャラグラで比較的オーソドックスなゲームを制作し，そのアルゴリズムを解説してみたいと思う。では，次号もよろしく。

コラム　ピコピコC言語リファレンス

COLUMN 3

今回から読み始めた人のために，前回学習したことを簡単におさらいしてみよう(今回解説した箇所は，適宜省略してある)。なお，当講座ではCを実践的に学習するという性質上，Cの文法事項については，本当に最低限(以下？)のことしか説明できていない。したがって，前回にも述べたとおりに，1冊は辞書的な使い方ができる解説書の購入をおすすめする。

※elがどのような処理の流れをしているかは，作者であるBotchy氏によって書かれたドキュメントファイルの"EL¥elopeate"を一読すると理解できるだろう。

■コメント

コメントは，あとでプログラムを見たときにわかりやすいように，説明などを書いておくためのものだ。コンパイル時には無視される。「/＊」と「＊/」で括られた範囲がコメントとなる。

例：/＊ あなたはポン酢の"ポン"の意味を知っているだろうか？ ＊/

「//」だけで，そこから行末(改行)までをコメントとすることもできる。これは特に簡単なコメントをつけたり，プログラムを修正したい場合，一時的にその行だけを無効にしたいとき(デバッグとか)に便利だ。

例：//"ポン"はオランダ語で，柑橘系のようなすっぱいものを指す言葉だ。

※例文のコメントが，くだらなくて申しわけない(でも，もっとあるぞ，ペヤングの"ペ"の意味とか……)。

■文

変数の宣言や関数の呼び出し、代入文などの式には必ず文の終了を示す";"をつける必要がある。

・変数の宣言とデータ型

変数とは，値が変化する入れもののようなもので，その内部には，いろいろなデータを入れることができる。変数を使うためには，事前にデータ型(変数の値の大きさ)と識別子(変数名)を宣言しなくてはならない。

例：int a;

これで，int型の変数aが宣言される。もっともよく使われるのが，int型(32ビット整数，±約21億)とchar型(主に文字列，マイナス128～127)で，これに加えfloat型(32ビットの浮動小数点型で，実数を扱う)を覚えておけば，だいたいは間にあうはずだ。なお，"int a=0;"のように変数の宣言と初期化は同時にすることもできる。

・変数の適用範囲(スコープ)

変数には、扱える範囲というものがある。これを変数のスコープと呼ぶ。基本的には，関数内{ }で括られた範囲がその変数のスコープとなる。

・配列

変数宣言の際に，変数の後ろに[ ](添字)を加えることによって，ひとつの変数で指定した添字の数マイナス1の数値を扱うことができる。

例：int a[10]={0,1,2,3};

これで配列には，a[0]～a[9]分の配列が確保され，a[0]には0，続いてa[1]には1が……というように値が入ることになる。

例：char b[10]="Hi!";

b[0]から順番に値(キャラクタコード)が入り，最後のb[3]に文字列の最後を示す「Null (ヌル)」(具体的には文字コードの0x000)が入る。つまり，文字列を扱うためには，「必要な文字数−1」分の領域の確保が必要となる。

・演算

Cでの演算方法は，"+"で加算，"-"で減算，"＊"で乗算(掛け算)，"/"で除算(割り算)，そして"%"で乗余算(割った余り)を行う。

例：a=a+1;

この例では，変数aに1を足した数が，aの中に代入されている。また，上記のように1だけ足したり，引いたりしたい場合には，a++;(インクリメント)やa--;(デクリメント)がよく使われるが，これは後述する繰り返しのforや，キャラクタの移動などによく利用される。

また=の後ろの変数が共通な場合には，a+=1;とすることも可能である。これらの演算は，状況に応じて使い分けるといいだろう。

■プリプロセッサ制御文・#include,#define

#で始まるプリプロセッサ制御文は，コンパイラがプログラムをコンパイルする前に処理される命令である。指定したファイルを取り込んだり，文字列の置き換えを行う。このプリプロセッサ機能を生かすことによって，ソースリストを簡潔に，読みやすいものにすることができる。

・ファイル取り込み

#include"el.h"

これでelのヘッダファイルが読み込まれることになる。この中には，Cの標準関数のヘッダファイル(stdio.h)などが含まれているので，ヘッダファイルの読み込みはこれだけで大丈夫だ。

・マクロ定義

#define abc 1

#defineは，後ろに指定された数値，文字列の置き換えを行う。#define abc 1となっているが，これ以後は，abcと記述すれば，1という数値に置き換わる。したがって，"a=abc;"とすると，aの値は1となる。

■制御構造

・条件判断・if

これはなんらかの条件が正しいかどうかを判定して，それらに応じて処理をしたい場合に使うものだ。

例：↓条件
```
if (a==0) {
処理
}
```

もし変数aの値が0なら，条件が真(正しいということ)になり，if以後の{ }で括られた範囲の処理が実行される。処理がひとつの場合，if文のすぐ後ろに書くことができるが，プログラムを見やすくするという意味で，あまり多用しないほうがいいだろう(変数の代入のような短い処理を除いて)。

例にあるとおり，その値が同じかどうか判定するには"=="を使い，逆に違うことを判定する場合には"!="を使う。

次に，大小の判定であるが，これは単純に">"，"<"でよい。注意してほしいのは">="，"<="で，"="を含む場合には，大小記号を"="の前につけることだ。

・繰り返し，forループ

forは，一定回数処理を繰り返したい場合に使用する。

例：
```
for(i=0;i<3;i++){
        処理
}
```

変数iを0にして，変数iが3より小さい間，変数iに1を加算して{ }内の処理を続ける。カッコの中を説明すると"(初期値;forを繰り返す条件;繰り返すためのカウント)"となる。

繰り返すためのカウントとして，「演算」で説明したひとつだけ変数の値を加えるインクリメントをしているのがわかるだろう。

■sprintf

例：sprintf(a,"%d",b);

sprintfは変数の数値を文字配列に変換する関数だ。

上の例では，aという文字配列に変数bの数値を入れている。真ん中の「%d」は変換指示記号で，変換する変数bの値を普通の数値(10進数)に直し，文字配列aに代入するという意味だ。変換指示記号の一覧を以下に記す。

%d：10進数変換
%x：16進数変換
%u：符号なし10進数変換
%c：1文字変換
%s：文字列変換。

ただし，必ず最後にNullを含んでいないといけない

・キー判定で使われるWin32 APIの仮想キーコード一覧

VK_UP,VK_DOWN,VK._LEFT,VK_RIGHT ……カーソルキー
VK_ESCAPE……エスケープキー
VK_SPACE……スペースキー
VK_RETURN……エンターキー
VK_NUMPAD0～VK_NUMPAD9……テンキー

※一般のASCIIコードで表現できるAや1は，そのまま'A'と'1'という風に使える。詳細はWin32APIのヘルプを参照のこと。

NetBSD for X68K 導入完全マニュアル

箕浦 真 Minoura Makoto

X68000系のマシンで動作するUNIX環境として知られるNetBSD。UNIXを勉強するなら最良のもののひとつとしても有名です。UNIXフリークの方以外でも，Human68k以外のOSに触れてみることは有益でしょう。今回はCD-ROMにNetBSD for X68Kを収録するとともに，導入の手順を徹底的に解説します。

第1部 概要編

世間を騒がすLinux

一昨年(1998年)あたりからLinuxブームともいうべき現象が起きています。現在では書店のコンピュータ関連のコーナーに行けば，タイトルにLinuxを冠した雑誌や書籍が山とあることに気づくでしょう。大手のコンピュータメーカーでも，昨年から競ってLinuxをサポートしはじめました。

Linux(*1)とは，Windows 98やMacOS，Human68kのようなOS(Operating System)の一種で下記のような特徴を持ちます。

- マルチユーザー：複数のユーザーがひとつのシステムを共有できます。複数のユーザーで，さまざまな資源を安全に共有するための機構を持ちます。ひとりで利用する場合も，通常の作業時と管理操作時など，ロール(役割)に応じて異なるユーザーとなることで，安全性を高めます。
- マルチタスク：複数の作業を並行して実行することができます。
- 高度なメモリ保護機能：あるプログラムが暴走したりしても，同時に動いているほかのプログラムやシステム自身に影響を与えません。したがって，システムは非常に堅牢です。
- 仮想メモリ機能：搭載しているRAM容量以上のメモリを要求するプログラムも実行できます。
- 高度なネットワーク機能：TCP/IPによるネットワーク機能を標準で持ち，インターネットのさまざまなサーバの仕事，たとえば，WWWサーバ，メールサーバソフトウェアなどを実行できます。

これらの特徴は，Unixと呼ばれる一群のOSと同じものです。実は，Linuxは，

- Unixとの互換性：Unixと高い互換性があり，Unix向けのアプリケーションは，最低限の修正で再コンパイルして利用することができます。

という点を最大の特徴のひとつとします。さらに，UnixとUnixサーバ/ワークステーションが，パソコン用OSと比べて一般に非常に高価であるのに対し，

- PC/AT互換機で動作：Linuxは，ありふれたPC/AT互換機(いわゆるDOS/Vマシン)のために開発されました(現在では，AlphaやSun SPARCstationなどにも移植されています)。
- (ほぼ)無償：Linuxはもともと，その大部分がGNU General Public License(GPL)で保護されており，本質的に無償です。

という点も大きな特徴です。GNU GPLは，フリーソフトウェアのためのライセンスですが，単に無料という意味ではなくfreeという言葉の本来の意味，つまり「自由」を前面に出したライセンスです。

- フリーソフトウェアである：GPLで保護されているため，誰もそのソースコードの再配布・改変などを妨げることはできません。

Linuxは，安価に構築できるインターネットサーバとして，またマイクロソフトの市場支配に対抗できる新しい勢力として注目を浴びています。また，重いGUIがオプションであることもあって，古い非力なマシンの再生という点でも重宝されています。

(*1)Linux：Linuxは本来，OSのカーネルのみを指す言葉ですが，ここではコマンド類や動作に必要なさまざまなプログラムを含めたOS全体を指すものとします。

PCで動くUnix!

実は，Linuxと似たような，フリーなUnix互換OSはほかにもいくつもあります。これらは，PCで動作するUnix互換OSという意味で，PC-UNIXと称されています。いずれも上記で述べたLinuxの特徴のほとんどを共有します。Linuxも含めて，得手不得手はあるものの，どれが優れたOSで，どれが機能が低いということはいえません。

- FreeBSD(*2)：PC/AT互換機，PC-98およびAlphaベースのワークステーションで動作。www.yahoo.comといった大規模サイトの運営にも利用されています。
- NetBSD(*3)：FreeBSDとは双子の兄弟ともいえるOS。Linuxを超える20以上ものアーキテクチャに移植されています。
- OpenBSD(*4)：NetBSDから派生したOS。高度なセキュリティ機能を統合しているのが特徴です。

これらは，名称にBSDというsuffixがつくことからわかるように，かつてのBerkeley(バークレー)版UNIX(BSD UNIX)(*5)の末裔です。その大部分は，GPLよりさらに自由度の高いBerkeley型と呼ばれるライセンスで保護されています。

PCで動作するUnixとUnix互換OSには，このほかにSun SolarisやBSD/OSなどもありますが，これらはフリーとはいえない(*6)ので割愛します。

(*2)FreeBSD：http://www.jp.freebsd.org/
(*3)NetBSD：http://www.jp.netbsd.org/
(*4)OpenBSD：http://www.openbsd.org/
(*5)BSD：当初，Unixの改良版として，UCB(University of California，Berkeley，カリフォルニア大学バークレー校)で研究・開発されていたOS。現行インターネットプロトコル(IPv4)の開発プラットフォームであったことは有名。AT&T分割とともにUnixのライセンスが厳しくなったことから一部コードの書き直しが行われ，UCBによる最終バージョン4.4BSDは，Unixのライセンスを必要としないフリーなUnix互換OSとなりました。
(*6)フリーとはいえません：コラム参照。

最強の移植性を誇るNetBSD

ここで注目したいのがNetBSDです。NetBSDは，そのプロジェクトの目標として，システムを“extremely portable”(*7)にすることを掲げています。移植性を高めるため，コード構成や各種フレームワークの設計・実装などに並々ならぬ努力を払っており，PC以外にも20を超えるプラットフォームに移植されています。

表1をご覧ください。懐しい名前もいくつかあるのではないでしょうか。ほら，特にいちばん最後。要するに，これまでの長い長い前振りは，すべて「開発もメーカーサポートも終わったobsoleteなOS(＝Human68k)なんか捨てて，NetBSDを使って，IPv6やJavaのような最新技術，Mozilla(*8)のような最新のソフトウェアを使ってみませんか」という一文がいいたいがためのものだったわけです。

(*7)extremely portable：http://www.jp.netbsd.org/Misc/about.html
(*8)Mozilla：http://www.mozilla.org/

NetBSD/x68kのススメ

旧Oh!Xの休刊前の最後の号で少し紹介されていたのを覚えている方もいらっしゃるでしょうか。NetBSD/x68kは，現在もしつこく開発が続けられています。

NetBSD/x68kは，X68030の発売直後の1993年，lndrvやSXkitの開発者としても知られる沖勝氏によって移植が始められ，1995年に公式にNetBSDのソースツリーに統合されました。各種アクセラレータ，純正およびMach-2 SCSI，Neptune-Xネットワークインタフェイス，内蔵シリアルポートなどをサポートしています。

最初に述べた特徴と重なりますが，NetBSDをHuman68kと比較してみましょう。

- ●インターネットとの親和性。現行インターネットのさまざまな技術の開発に使われたOSの子孫であり，次世代インターネット(IPv6(*9))の開発プラットフォームとしても現役です。もちろん，ネットワークを通じたリモート管理なども可能です。
- ●豊富なソフトウェア。Unix互換OSですから，各種Unix向けソフトウェアがそのまま，またはわずかな修正で利用できます(*10)。これらのインストールを容易にする仕組みであるpackage機構も用意されています。
- ●最新の開発環境。gccをはじめとする最新の開発環境がすぐ利用できます。
- ●堅牢性。ソフトウェアのバグで白窓が出て，以降システム全体が不安定になったりすることはありません。
- ●システムのソースコードが公開。OSやプログラミング，プロトコルの学習に，バグ対策に。NetBSDは，PC-UNIXの中でももっともコードが綺麗だといわれています。

半面，このような欠点もあります。

- ●Human68kとは異なり，スーパーバイザモードでハードウェアを直接叩いたりすることはできません。が，そのハードウェアに対応するデバイスドライバさえ作ればいいわけですし，それもシステム全体のソースコードが公開されているおかげで比較的容易です。
- ●上記と関係しますが，X680x0特有の機能が使いにくいです。スプライトをはじめとする画面周りの特殊機能はほとんどサポートされていません。これもデバイスドライバ次第なわけですが。
- ●Human68kやSX-Windowのソフトウェアが実行できません。一時期沖氏らによってHuman68kエミュレータ「ひゅう真似(Humane：Human68k Emulator)」が開発されていましたが，現在はメンテナンスされていません。
- ●X680x0のすべてのモデルで動作するわけではありません。
- ●ドキュメントの大半は英語で，それらの日本語訳はわずかしかありません。

表1 NetBSDの動作するプラットフォーム('99年11月1日現在)

machine	machine arch	Release/snapshot	Note
alpha	alpha	1.4.1	AlphaPC/WS/サーバ
amiga	m68k	1.4.1	CommodoreAmiga
arm32	arm32	1.4.1	ARMプロセッサ利用のマシン群
atari	m68k	1.4.1	Atari
bebox	powerpc	スナップショット	BeBox
evbsh3	sh3	NEVER	SH3評価ボード
hp300	m68k	1.4.1	HP9000/300、HP9000/400
hpcmips	mipsel	スナップショット	MIPSベースWindowsCEマシン
i386	i386	1.4.1	PC/AT互換機
mac68k	m68k	1.4.1	68k Mac
macppc	powerpc	1.4.1	PowerMac
mmeye	sh3	NEVER	www.brains.co.jp参照
mvme68k	m68k	1.4.1	680x0ベースのVMEマシン
newsmips	mipseb	スナップショット	Sony RISC NEWS
next68k	m68k	1.4.1	NeXTCube
ofppc	powerpc	スナップショット	OpenFirmware搭載機
pc532	ns32k	1.4.1	同人ハードウェアの一種
pmax	mipsel	1.4.1	DECstationなど
sparc	sparc	1.4.1	Sun SPARCstationなど
sparc64	sparc64	スナップショット	Sun ULTRAシリーズ
sun3	m68k	1.4.1	Sun-3
vax	vax	1.4.1	DEC（現Compaq）VAX
x68k	m68k	1.4.1	X680x0

※このほか，NECのPC-98x1などでも動作する(Unofficial port)。
※スナップショットは，開発中のバージョンをバイナリで公開することで，開発者を中心としたユーザーの利便を図るためのもの。開発中のソースコードは常に公開されている(NetBSD-current)。

フリーソフトウェアとオープンソース運動 column

現在では，ソフトウェアの売買とは，ソフトウェアの収められたメディアを売買することではなく，ソフトウェアそのものを利用するための権利の売買であるという認識が定着しています。市販のソフトウェアを購入すると，利用者の権利などを定義した利用許諾契約(ライセンス)が同封されており，インストールする前に契約への同意が求められます。

このライセンスには製品やメーカーによってさまざまな形態がありますが，一般によく知られているもののひとつとしてGNU(http://www.gnu.org/)の一連のソフトウェアなどに適用されるGPLがあります。GNU GPLとは，GNU General Public License(GNU一般公有使用許諾)の略で，原文をhttp://www.gnu.org/copyleft/gpl.htmlで，日本語訳をhttp://www.sra.co.jp/public/doc/gnu/gpl-2j.txtでそれぞれ閲覧できます。その骨子は「1. ソースコードは自由に複製や再配布が可能である。2. 改変したソースコードも，GPLに従う限り自由に再配布が可能である。3. バイナリコードを配布する場合，ソースコードも入手できるようにしなければならない」などです。

LinuxやNetBSDを含めたUnixの世界で“Free”といえば，無料という意味ではなく「自由」という意味を指すことが大半です。つまり，Free softwareとは，入手や再配布のほか，ソフトウェアの改変や改変したソフトウェアの再配布，コードの流用なども自由にできるソフトウェアのことです。GNU GPLによるソフトウェアは，free softwareの典型例といえるでしょう。NetBSDのライセンスは，copyrightの記述を残すことで，ソースコードもバイナリコードも自由に配布・改造ができるというもので，もちろんこれもフリーソフトウェアといえるでしょう。

一方，英語の“Free”には，もとの「自由」という意味から派生して，「無料」という意味もあります。Free softwareが，freely downloadable software，つまりネットワークを通じて自由にダウンロードできるソフトウェア，などの意味で使われることも多くなってきました。特にパソコンの世界では，有料のsharewareに対する言葉として使われています。こうした用語の混乱を避けるため，新たにopen source softwareという言葉が考え出されました(http://www.osi.org/)。

すると今度は単にソースコードを公開するだけで，改変には制限のあるソフトウェア，たとえばSunのJDK(Java Development Kit)などもopen sourceといわれるようになってしまっています。言葉というものは，難しいものです。

こうした比較結果からは，用途に応じてHuman68kとNetBSDの使い分けをすることによって，より幸せになれる，という結論が導かれると思います。

(*9) IPv6の実装：現行の最新版であるNetBSD-1.4.1では未サポートですが，近々公開される次期バージョン1.5では，Kameプロジェクト(http://www.kame.net/)によるIPv6の実装が取り込まれています。Kameプロジェクトによる実装は，IPv6の実験ネットワークである6bone(http://www.6bone.net/)での稼働実績でももっとも信頼を得ているもののひとつです。

(*10) そのまま～利用できます：Unixの世界では，多くのソフトウェアがソースコードで配布されます。

以降の記事では，X680x0に実際にNetBSD/x68kをインストールし，インターネット接続をするところまでを解説します。

第2部　インストール編

さて，今回本誌付録CD-ROMに，NetBSD/x68kの最新版を収録していただけることになりました。これを利用したNetBSD/x68kの導入方法を紹介します。

インストールの前に

インストール方法は，付録CD-ROMの¥NETBSD¥X68Kディレクトリに，テキスト(改行コードがUnix形式なので，Windowsのメモ帳などではすべての行がつながって見えてしまうので注意)，HTML，PS(ポストスクリプト)，more(Unixのコマンドmoreのための制御コードを含むテキスト)の4つの形式で収められています。手持ちのWWWブラウザなどで一読しておきましょう。ただし，英語です。

まずはハードウェアの準備をします。実はここがいちばん敷居が高かったりするのですが，簡単なところから順番に……。

・メモリ

メモリは最低4MBを必要とします。4MBメモリの場合には，専用に作られたSMALLカーネル(後述)をインストールする必要があります。できれば12MBは実装しておきたいところです。

なお，九十九電機製のX68030用増設メモリTS-BE16や，060turboのオンボードSIMM増設メモリにも対応しています。この場合も本体には4MB以上のメモリを実装しておいてください。

・ハードディスク/CD-ROMドライブ

SCSIハードディスクのみをサポートしています。また，インストーラがディスク全体を占有しようとするため，事実上専用にディスクを1台用意する必要があります。120MBのMOにもインストールできなくはありませんが，かなり窮屈(かつ遅い)でしょう。

また，本稿では付録CD-ROMからインストールすることを前提としておりますので，X680x0にCD-ROMドライブを接続し，またHuman68kがNetBSDのインストーラを読めるようCD-ROMドライバをインストールすることが必要です。

なお，SCSIホストアダプタとしては，X68000 SUPER以降に内蔵のもの，純正のSCSIボード，Mach-2をサポートしています。

・CPU

さて，最後に回したのがいちばん重要なアイテム，CPUです。

まず，NetBSDの高度な仮想メモリ機能をサポートするために，MMU(メモリ管理ユニット)が必須になります。よく知られているように，X68030のCPUは，MC68030からMMU機能を削ったMC68EC030です。幸いにして，X68030の場合はCPUを(ECではない)MC68030に差し替えることで，MMUを利用することが可能になります。

また，X68000シリーズのCPUであるMC68000(HD68HC000)にはMMU付きのモデルはありませんし，MMUをあとから付加することもできません。この場合，MC68030以上のCPUを積んだアクセラレータを利用することになりますが，もっともポピュラーなアクセラレータXellent30のCPUもやはり68EC030です。したがって，この場合もなんらかの対策が必要です。

また，NetBSDに限らずUnix系OS一般で使われるGUIプラットフォームX Window Systemでは，X serverと呼ばれるプログラムがフレームバッファ(VRAM)を扱える必要があります。X680x0では，ハードウェア的にフレームバッファはスーパーバイザモードでしか操作できません。しかしながら，X serverは，ユーザーモードで動作するごく普通のアプリケーションプログラムですので，なんらかのハードウェア的な対策が必要です。こうした対策をした場合，Human68kなどで動作するユーザーモードアプリケーションも，フレームバッファをアクセスできるようになってしまうわけですが，動作していたアプリケーションが動作しなくなることはありませんし，またMMUの機能を用いてより柔軟な保護ができます(*11)ので問題はないでしょう。

なお，X serverは，NetBSD/x68kでグラフィックを表示するには事実上唯一の手段ではありますが，他機種の場合と異なり，日本語の表示手段としては必要がありません。非常に重い処理となりますので，「X serverを利用しない」という選択肢もあることを付け加えておきます。

・X68030シリーズの場合

前述のように，単にソケットに差さっているCPUをMC68030に交換するだけで大丈夫です。MC68030は，秋葉原の部品屋などで入手できるほか，パソコンショップ満開でも扱っているようですので，地方の方は利用してみてください。なお，MC68030にはパッケージやクロックによっていくつかの種類がありますが，オーバークロック改造をしていないX68030の場合は「MC68030RC25」というタイプを選んでください。

なお，CPUの交換には当然X68030を開ける必要があり，これは慣れないと非常に難しい作業です。開け方については，あちこちで紹介されていますので，ここでは触れません。

次に，X serverを動かす場合は，MC68030のFC2ピン(図1)をポッキリ折ってください。高価なCPUに傷をつけたくない場合，ソケットを挟むとよいでしょう。このソケットも満開で扱っているようです。

また，浮動小数点演算用のコプロセッサ(FPCP)MC68881またはMC68882をインストールすることが強く推奨されています。コプロセッサエミュレータが用意されてはいますが，不完全です。

・X68030+040turboの場合

040turboのCPUであるMC68040(XC68040)は，MMUとFPCPの機能(FPU)を内蔵しています。特に問題なく利用できます。

X serverを利用する場合，本体基板との接続部にあたるソケットのFC2ピンを折ることになります。この場合もソケットを挟むことで，040turboに傷をつけることなく対応可能です。

・X68030+060turboの場合

060turboのCPUであるMC68060(XC68060)は，MMUとFPUを内蔵しています。特に問題なく利用できます。

図1　MC68030のFC2ピン(底面図)

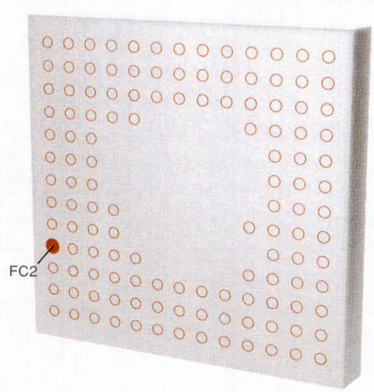
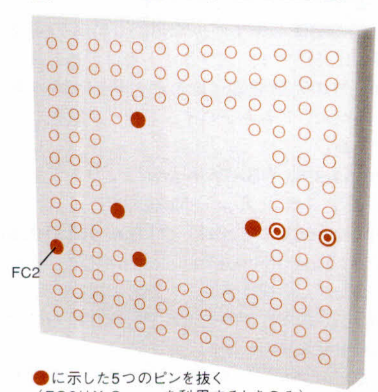


図2　MC68030のMMUDISピン



●に示した5つのピンを抜く
(FC2はX Serverを利用するときのみ)
◉に示した2つのピンを抵抗でつなぐ
(図中右側がMMUDIS，左がVcc)

X serverを利用する場合は，060turbo上のジャンパを切り替えることになります。これに関しては，060turboのマニュアルをご覧ください。

・X68000シリーズ+Xellent30シリーズの場合

Xellent30シリーズのCPUもMMUのないMC68EC030ですので，交換が必要となります。

MC68EC030とMC68030は，基本的にはピン互換なのですが，わずかに非互換な部分があり，Xellent30シリーズはこれに引っかかっています(*2)。具体的には，MC68030のみにあるピンのひとつ(MMUDIS)がどこにも接続されず，不安定な状態になっていますので，これをプルアップしてやります。図2に示す2つのピンを，4.7kΩ程度の抵抗で継いでください。また，図に示した余分なピンも切断しておきます。

X serverを動かすためにはX68030シリーズの場合と同様CPUのピンを折る必要があります。FPCPのインストールが推奨されるのも同様です。

・X68000シリーズ+Jupiter-Xの場合

Jupiter-XのCPUであるMC68040(XC68040)およびXC68060には，MMUとFPUを内蔵しておりますので，問題なく利用できます。

X serverを動かすための措置は，FTZオリジナル配布のα版を除き，必要ありません。060turboのジャンパ切り替えにあたる機能がソフトウェアで制御できるようになっています。

・その他のアクセラレータの場合

その他のアクセラレータで動作するかどうかは未確認です。040Excelでは動作しないとの報告があります。

(*1)MMUの機能を用いてより柔軟な保護が可能：詳しくは触れませんが，030SYSpatch(X68030/Xellent30)，040SYSpatch(040turbo)などを利用します。
(*2)Xellent30シリーズでは改造が必要：ロットやモデルによっては対策は必要ない，とのことです。

ディスクの接続順について

NetBSD/x68kでは，ハードディスクをSCSI ID順にsd0，sd1……という名で呼びます(sdはScsi Diskの略)。これは，若いIDのディスクの電源を切ったまま起動した場合と入れた状態で起動した場合とでは，番号が変わってしまうことを意味します。Human68kにおいても同様にドライブ文字がずれることがありますが，それと同様です。

インストール時には，各ドライブの電源は，普段NetBSDを使うときと同様の状態にしておいてください。操作ミスなどによる事故を防ぐため，インストール時にはディスクの電源を落としておきたい，という場合は，インストーラが起動して最初のプロンプトが出てから落とすようにします(SCSIバスが不安定になるのであまりすすめられることではありませんが)。

また，ディスクはあらかじめHuman68kのFORMAT.Xコマンドで「装置初期化」を行っておいてください。

ディスク利用の計画

さて，NetBSDは，ハードディスクを主にファイルシステムとスワップの2つの目的に利用します。

・ファイルシステム

通常のファイルを置きます。

ファイルは，DOS/WindowsやMacOSなどと同様，ディレクトリ(フォルダ)によるツリー構造に整理されます(というより，DOSやMacOSなどがUnixを真似た)。区切り文字には'/'を用い，「usrディレクトリの中のbinディレクトリのなかのgccというファイル」は，/usr/bin/gccのように表します。

DOSではドライブ(物理ドライブやパーティション)ごとに複数のツリーを用いて管理しますが，Unix系OSでは，すべてのドライブを単一のツリー上に管理します。通常起動ドライブがこのツリーの根の部分となり，「ルートファイルシステム」と呼ばれます。そのほかのドライブは，このルートファイルシステムの適当なディレクトリ(マウントポイント)に載っけることで，アクセス可能になります。この操作をマウント(mount)と呼びます。

伝統的には，1台のディスクであっても，ルートファイルシステムのほかに，/usrファイルシステム，/varファイルシステム，/homeファイルシステムなどのパーティションに分けることが多いようですが，ここでは全体を単一のファイルシステムに収めてしまうことにしましょう。この方法には，パーティションごとに空き容量に偏りが起きることがないという利点があります。

なお，X680x0では1台のディスクを15のパーティションに分けることができますが，NetBSD/x68kではこのうち先頭の7パーティションのみを利用できます。それぞれa，b，d，e，f，g，hとアルファベットを振り，たとえば1台目のドライブであれば，sd0a，sd0b……と呼びます。Cパーティションは，ディスク全体を指す特殊なパーティションとして扱われます。ルートファイルシステムは通常aパーティションに置きます。

・スワップ

仮想メモリシステムでは，実際に搭載しているRAM容量より多くのメモリがあるかのように見せることが可能です。RAM容量を超える部分についてもどこかに記憶しておく必要があるわけですが，このための記憶装置を仮想メモリのバッキングストア，またはスワップと呼び，通常ハードディスク上に取られます。

Windows9xでは通常，C:¥WINDOWS¥WIN386.SWPというファイルがそれにあたりますが，NetBSDでは通常，「ルートファイルシステムと同じディスクの2番目のパーティション」(すなわちbパーティション)が用いられ，これをスワップパーティションと呼びます(*3)。

スワップパーティションの大きさは，伝統的にはRAM容量の2倍から3倍が適当だとされています。この場合，仮想メモリの大きさは，RAM容量の3倍から4倍，ということになります。

(*3)スワップパーティション：スワップはファイルに取ることも可能です。

以上を理解したうえで，ここでは手元にあった2GBのSCSIハードディスクを，「36MBのスワップパーティションと，残りの約2GBのルートファイルシステム用のパーティション(ルートパーティション)」に分けて利用することにします。

インストーラの起動

まずはHuman68kを起動します。CD-ROMを認識していることを確認してください。以下CD-ROMドライブがF:であると仮定します。

NetBSD/x68kのインストーラは，付録CD-ROMの¥NETBSD¥X68K¥INSTALL¥MISCディレクトリに収録されていますので，ここにカレントディレクトリを移します。ファイルNETBSDがインストーラ，LOADBSD.Xがインストーラを起動するためのコマンドです。

```
A> F:
F> cd \NETBSD\X68K\INSTALL\MISC
F> loadbsd netbsd
```

うまくいけば，いったん画面が消え，NetBSDの起動メッセージが表示されるはずです。しばらくは接続されているデバイスの検出を行っている旨の表示があるはずです。NetBSDの世界へようこそ。

Unixのオンラインマニュアル column

本文の説明には，「hoge(1)」などという表現が出てきます。これは，Unix系OSのオンラインマニュアル(man page)の一項目を示す表現です。man pageは，いくつかのセクションに分かれており，NetBSDの場合は1から9までの9セクションからなります。セクション1が一般コマンド，5がファイル形式，8が管理用コマンド，などと分類されており，「hoge(1)」は通常hogeという一般コマンドのman pageを指すほか，そこから派生してhogeコマンドそのものを指すこともあります。参照にはman(1)を用いてman 1 hogeなどとします。NetBSDをインストールしたら一度試してみてください。もちろんman(1)自身もman 1 manで参照することができます。

なお，オンラインマニュアルの形式としては，man pageのほかにGNU infoという形式がよく利用されています。Info形式のオンラインマニュアルを参照するには，NetBSDのコマンドプロンプトからinfoと入力してみてください。

うまくいかない場合は……

- ●CPUは正しいものに切り替わっていますか? MC68000やMC68EC030では動作しません。
- ●CONFIG.SYSにSHAREの設定がありますか? ない場合，SHARE=40 20などの記述を追加してください。
- ●CONFIG.SYSから，上記SHAREやCD-ROMドライバなど最低限の記述を除いて削除してみてください。
- ●それでもだめな場合，本稿の最後に述べるメーリングリストx6netbsd-usersなどで相談してみてください。その際，ハードウェア環境，Human68kの環境などを詳しく付記してください。

ディスクのパーティショニング

さて，しばらくすると，installかupgradeかを選ぶメニューが出てきますので，まずは迷わずinstallを選んでください。

このあとは，画面の指示に従って質問に答えていくことで，ディスクのパーティショニングを行います。質問の各最後の[]内の文字列が，デフォルト値として単にリターンキーを押した場合に採用されます。質問は英語ですが，さほど難しい質問はないと思います。わかりにくいものについて説明しておきましょう。

```
- What kind of disk is it? (one word please)
[mysd]
```

これはあまり意味がある質問ではありませんので，デフォルトどおりmysdと答えておいてください。

```
- View the boot messages again? [n]
```

NetBSDのファイルシステム形式ffs(First File System)は，ディスク上のデータが同心円状に記録されていることを利用して，シーク時間を抑え，データを高速に検索したり読み書きしたりできるよう工夫されています。このため，ディスクのセクタサイズ，ヘッド数などのジオメトリ情報を入力する必要がありますが，これらの情報は起動時にディスクの検出を行っている際に表示されています。これを参照するか，という質問です。

ここで「y」を選ぶと，more(1)で起動メッセージが表示されます。先頭部分は'^@'というゴミが表示されていますが，しばらくスペースキーを押していると，起動メッセージが出てくるはずです。moreコマンドの使い方は，Human68kやMS-DOSと同じですが，Bキーで逆スクロールすることができます。

ちなみに手元にあるディスクでは，

```
sd0 at scsibus1 targ 1 lun 0：<SEAGATE, ST324
30N, 0170> SCSI2/direct fixed
sd0：2049MB, 3992 cyl, 9 head, 116 sec, 512
bytes/sect x 4197405 sectors
```

と表示されました。したがって，ブートメッセージを参照したあとの問いには以下のように答えます。

```
Number of bytes per disk sector? [512] -> 512
Number of disk cylinders? -> 3992
Number of disk tracks(heads)per disk cylin
der? -> 9
Number of disk sectors per disk track? -> 116
```

なお，セクタサイズの入力も求められますが，512バイト/セクタ以外のディスクは実はサポートされていません。また，最近のディスクはディスクの内周部と外周部ではセクタ数が異なるなど複雑な構成をとっており，起動時に表示されるジオメトリ情報はダミーであることが多いようです。さらにNetBSD/x68kでは，そもそもここで入力された情報は(現在のところ)利用されません。

```
- What units would you like to use?[cylinders]
```

上記のような高速化のため，ファイルシステムのサイズはシリンダ単位に指定します。しかしながら，シリンダあたりの実際のバイト数はディスクによって異なるなど，計算が面倒ですので，セクタ単位での指定も許すようになっています。その場合，実際に作られるファイルシステムのサイズはシリンダ単位に切り下げられ，余った部分は無駄になります。

このあと実際に各パーティションのサイズを入力していくことになるわけですが，先ほど決めたようにswapパーティションに36MB割き，残りは単一のファイルシステムとする場合，このように計算します。

1 cylinder = 1044 sectors = 534528 bytes
(起動時の表示より)
36MB = 37748736 bytes ≒ 71 cylinders
(37748736/534528)

したがって，下記のように入力することになります。

```
Your disk has a total of 4167648 512 byte
sectors,
arranged as 3992 cylinders which contain 1044
sectors (534528 bytes) each.

You can specify partition sizes in cylinders
('c') or sectors ('s').
What units would you like to use? [cylinders]
c

There are 3991 cylinders left to allocate.

Root partition size (in cylinders) ? 3920

71 cylinders remaining in NetBSD portion of
disk.
Swap partition size (in cylinders) ? 71
```

ディスクが3992シリンダ持っているにもかかわらず，3991シリンダしか残っていないと表示されているのは，ディスク先頭のブートブロックやパーティションテーブルなどに64セクタほど取られているせいです。

ここまでの過程では，ディスクに実際に書き込むことはありません。もし途中で入力を誤ったりした場合は，あわてず^C(CTRLキーを押しながらCキーを押すことをこう表記します)を押し，

```
# install
```

と入力すればインストーラが再起動します。

```
- Are you SURE you want NetBSD installed on
your hard drive?(yes/no)
```

最後の確認です。祈りつつyesと入力しましょう。もう後戻りはできません。

GOOD LUCK!と表示されたら，パーティショニング(およびファイルシステムの初期化)は終わりです。

システムの展開

次に，CD-ROMからシステムをハードディスク上に展開します。

まず，CD-ROMをmount(8)します。

```
# mount -o ro -t cd9660 /dev/cd0a /mnt2
```

この操作で，CD-ROMの内容が/mnt2以下にマウントされます。mount(8)に関しては，あとでオンラインマニュアルをチェックしておきましょう。

cd0aが，CD-ROMのデバイス名です。1台目から0，1……と番号が振られること，また'a'が最初のパーティションを表すのはハードディスクと同じです。もっとも通常CD-ROMをパーティション分割することはありませんが。

システムは，以下の12の'set'に分割されています。必要なものを選択します。

- ●base　基本的なコマンド群など。必須です。
- ●etc　基本的な設定ファイル群。(旧バージョンからのアップグレードではない)新規インストールの場合は必須です。
- ●kern　NetBSDのカーネルです。カーネルとは，MS-DOSのIO.SYS，Human68kのHUMAN.SYSなどに相当する，OSの核となる部分です。必須ですが，RAMが4MBしかない場合は，代わりに後述のSMALLカーネルを利用します。
- ●man　base setのためのオンラインマニュアルです。インストールするべきでしょう。
- ●comp　開発キットとそのオンラインマニュアルです。C，C++，Objective C，Fortran77の各コンパイラやアセンブラなどが含まれます。
- ●text　各種テキスト処理プログラムが含まれます。
- ●games　各種ゲームです。
- ●misc　その他のドキュメントなどです。
- ●xbase　X Window Systemの基本的なコマンド群などです。
- ●xcomp　Xの開発キット。
- ●xserver　X serverです。
- ●xfont　Xで利用されるフォントファイル群。X serverを利用しない場合は必要ないでしょう。

展開にはExtractというコマンドを用います。

```
# Extract base
What directory should be used to find and/or
store installation
files such as base.tgz and kern.tgz? [/mnt/
usr/distrib] _
```

この質問は，最初のsetの展開のときのみ表示

されます。ここでは，/mnt2/netbsd/x68k/binary/sets と答えます（ファイル名は，Human68kからは大文字で見えましたが，NetBSDからは小文字に見えます）。

以下，必要なだけExtractを実行します。順序は任意でかまいません。

```
# Extract kern
# Extract etc
# ...
```

最後に，RAMが4MBのシステムにインストールする場合は，kern setの代わりにSMALLカーネルをインストールします。

```
# cd /mnt
# gzip -dc /mnt2/netbsd/x68k/binary/kernel/small.gz > netbsd
```

以上でシステムの展開が終了しました。

最後に，忘れずにConfigureを実行し，再起動します。

```
# Configure
...
What is this machin's hostname? [unknown.host.domain] myname

What DNS domain is this machine in (this is NOT its YPdomain name)? [myname] my.domain
...
# reboot
```

hostnameとdomainを入力するよう求められます。ネットワークに参加する場合は，管理者の指示どおりに入力してください。ネットワークを利用しない場合や，一般のISPを経由してインターネットにPPPで接続する場合は適当でかまわないでしょう。Neptune-Xを装備している場合には，IPアドレスなどの入力も求められます。

最後に，SCSIホストアダプタとしてMach-2を利用していない場合，適当なブートセレクタを導入して，NetBSDをインストールしたディスクから起動できるようにしましょう(*4)。Mach-2の場合は起動時にCTRL+Aを押すことで，起動ディスクが変更できます。以降の設定法は，次の「設定編」で解説します。

(*4) ブートセレクタ：SWITCH.Xで起動ドライブを変更してもよいのですが，NetBSDからは設定変更ができないため，再度別なディスクから起動したい場合は，一度フロッピーからHuman68kを起動し，SWITCH.Xを実行し直す必要があります。NetBSD-1.5からはSRAMのメモリスイッチの設定をするmemswitch(8)が用意される予定です。

サポート

最後に，NetBSDを利用していくにあたって重要な情報の入手先を列挙しておきます。NetBSD/x68kを使っていくうえでは，port-x68kおよびx6netbsd-users，創夢のメーリングリストを購読しておくことを強くおすすめします。

・WWW上の情報源

●The NetBSD Project
http://www.netbsd.org/
公式サポートサイト。NetBSDに関するあらゆる情報が集積されています。国内にミラーがありますので，日本からはできるだけ下記ミラーを利用してください。

●日本NetBSDユーザーグループ（JNUG）
http://www.jp.netbsd.org/
上記のミラーのほか，日本語訳，ユーザーグループの情報，国内の情報，ほかの関連サイトへのリンク集などもあります。

●NetBSD/x68k
http://www.rd.cc.kshosen.ac.jp/~kamahara/NetBSD-X680x0/index.j.html
最近更新が止まっているようですが，下記メーリングリストに関する情報などもあります。

・メーリングリスト

●port-x68k
http://www.jp.netbsd.org/ja/MailingLists/index.html#port-x68k
The NetBSD Projectが運営する，NetBSD/x68kに関するメーリングリスト。英語が中心。

●本家のメーリングリスト
http://www.jp.netbsd.org/ja/MailingLists/index.html
The NetBSD Projectは，そのほかにも数十のメーリングリストを運営している。

●x6netbsd-users
http://www.rd.cc.kshosen.ac.jp/~kamahara/NetBSD-X680x0/mailinglist.j.html#USERS
NetBSD/x68kユーザーのためのメーリングリスト。日本語が使える。

●x6netbsd-develop
http://www.rd.cc.kshosen.ac.jp/~kamahara/NetBSD-X680x0/mailinglist.j.html#DEVELOP
NetBSD/x68kの開発に関する議論のためのメーリングリスト。

●NetBSDメーリングリスト
http://www.re.soum.co.jp/~jun/welcome.html#netbsd
通称創夢のメーリングリスト。NetBSD一般について扱う。

●JNUGのメーリングリスト
http://www.jp.netbsd.org/ja/JP/ml.html
日本NetBSDユーザーグループによるメーリングリストが列挙されている。

・NetNews

●fj.os.bsd.netbsd
NetBSDに関するグループ。

日本NetBSDユーザーグループ　column

日本にはどういうわけかNetBSDユーザーが非常に多いらしく，NetBSDに関連した本が山ほど出版されていたり，開発者が異様にたくさんいたりします。

日本NetBSDユーザーグループ（JNUG：Japan NetBSD Users' Group）は，The NetBSD Projectの成果を日本向けに提供すること，日本でのNetBSDに対する成果の還元を促すこと，NetBSDユーザー，開発者相互の交流などを目的として，1999年8月に設立されました。筆者もその運営委員会に参加しています。現在の主な活動は，www.jp.netbsd.orgをはじめとするサーバ群の運営や，ドキュメントの翻訳が中心となっています。特に，NetBSDのソースコードの任意のファイルの，初期から最新版にいたる任意のバージョンを，更新履歴とともにWWWを用いて閲覧することのできる，cvswebサーバ（http://cvsweb.netbsd.org/）を，世界に先駆けて提供しています。

JNUGではこうした日常活動のほか，さまざまなイベントに参加してNetBSDのアピールをしています。機会があれば参加してみてください。

本稿でも紹介されているJNUGのWWWサイトには，会員登録用のフォームが用意されており，誰でも自由に会員になることができます。イベント情報なども会員のメーリングリストでは必ずアナウンスされますので，NetBSDのユーザーの皆さんはぜひとも会員になってみてください。

また，ドキュメント翻訳プロジェクトの協力者も募集しておりますので，ガッツとボランティア精神にあふれた方は，お気軽にwww-changes-ja@netbsd.orgにコンタクトしてみてください。

第3部　設定編

設定編とはいっても，NetBSDという大きなシステムすべての設定法を1からすべて解説すると，本が何冊も書けてしまいます。ここでは，NetBSDやNetBSD/x68kに特有の事項を中心に基本的な部分を説明し，その他の部分は市販のUnix入門書やオンラインマニュアルを参照していただくという方針でいきたいと思います。重要な部分を端折っていたり，全体に飛ばし気味だったりしますが，どうかご容赦ください。

インストールの完了

さて，「インストール編」でNetBSDをインストールしたディスクからX680x0を起動すると，こんなメッセージとともに起動が中断すると思います。

```
/etc/rc.conf is not configured. Multiuser boot aborted.
Enter pathname of shell or RETURN for sh : _
```

ここでリターンキーを押すと，

```
Terminal type?
```

と聞かれますので，x68kと答えてリターンキーを押します。

```
Don't login as root, user the su command.
# _
```

と，コマンド待ちの状態になります。この状態を，シングルユーザーモードと呼び，特殊な管理操作の際に利用されます。ここで，インストールの最後の仕上げをしましょう。

具体的には，ファイル/etc/rc.confを編集します。BSDの標準エディタといえばviです。NetBSDを使うからには皆さんにもぜひvi(*1)を覚えていただきたいわけですが，いかんせんこいつは慣れるまでが大変なエディタです。ここでviの解説を始めるとそれだけで1冊本が書けてしまいますので，ここではとっつきやすいエディタを紹介し，特別にインストール法をお教えすることでお

許しください。

シングルユーザーモードでは，ファイルシステムが読み込み専用になっていますから，まずはこれを書き込めるようにマウントしなおします。

```
# mount -u /
```

次に付属CD-ROMをマウントします。

```
# mount -o ro -t cd9660 /dev/cd0a /mnt
```

CD-ROMからjoeというエディタをインストールします。

```
# cd /mnt/netbsd/ohx
# pkg_add joe-2_8.tgz
```

これでjoeというエディタがインストールされました。joeの使い方は，^K H(CTRLキーを押しながらKを押し，続いてHを押す)で表示されます。

CD-ROMは，マウントするとロックされます。取り出すには，一度アンマウントという操作を行います。

```
# cd /
# umount /mnt
# eject /dev/cd0a
```

さて，インストールしたエディタを利用して/etc/rc.confを編集します。

```
# joe /etc/rc.conf
```

カーソルキーを利用してカーソルを移動し，12行目前後にある，

```
rc_configured=NO
```

という行を，

```
rc_configured=YES
```

と書き換えます。必要であれば，rc.conf(5)を参照し，以下の行も編集しておいてください。

これで，最低限の設定は完了です。^Cを押してjoeを終了し，編集結果を保存するか否かの問いに‘Y’と答えます。再度reboot(8)すると，今度は，

```
NetBSD/x68k (myname) (console)
login:
```

というメッセージが出るでしょう。これが通常のオペレーションで利用するマルチユーザーモードです。

(*1)NetBSDに含まれるviは，実際にはviと「bug-for-bug compatible(バグまでコンパチ)」なnviがあります。

ユーザーの設定

NetBSDを利用するためには，まずユーザー認証を経て，人間の命令をコンピュータに伝えるためのプログラム「シェル」を起動することから始めます。これを，ログインといいます。

NetBSDはマルチユーザーOSですが，初期状態では管理者(rootと呼ばれる)のみが登録されています。login：プロンプトに対して，ユーザー名rootを入力しましょう。

```
login：root
Copyright(c)1996, 1997, 1998, 1999
The NetBSD Foundation, Inc.  All rights reserved.
...
Don't login as root, use the su command.
Terminal type? [x68k] _
```

Terminal typeはデフォルトのままのx68kでかまいませんから，リターンを押します。

まずはrootユーザーのパスワードを設定しましょう。

```
# passwd
Changing local password for root.
New password：_
```

パスワードは6文字以上127文字までの文字列です。あまり簡単なパスワードだとはねられますので，数字や記号，アルファベットの大文字などを交ぜてみてください。パスワードは入力してもエコーバックされません。確認のため二度聞かれますので注意してください。

rootユーザーは，その別名であるスーパーユーザーという言葉が示すように，非常に大きな権利を持っており，日常的に利用するには危険です。WindowsやHuman68kを利用していて，C:¥WINDOWSディレクトリやC:¥CONFIG.SYSのような重要なファイルを誤って消してしまってあわてたことはありませんか? NetBSDでは，システムにかかわるファイルをroot以外のユーザーが編集したり消したりすることはできませんので，こうした事故を防ぐことができます。このためには，日常の作業を行うためのユーザー情報(アカウント)を登録し，普段はそのユーザーとして利用することが必要です。

ユーザーの追加には，vipw(8)というコマンドを利用します。

```
# vipw
```

これはユーザー情報を管理するファイルpasswd(5)を編集するためのコマンドです。通常viが起動してしまいますので，まだviの使い方をマスターしていない場合は，

```
# EDITOR=joe vipw
```

として利用するエディタを指定してください。

passwd(5)は，1行1エントリで以下のような構造を持ちます。すでに登録されているのは，管理者rootのほか，daemon，operatorなどシステムが利用する仮想的なユーザーです。

```
minoura:*:1001:20::0:0:Minoura  Makoto:/home/minoura:/bin/sh
```

1エントリは，‘:’で区切られた10の項目からなります。

- 項目1(minoura)：ユーザーを識別するためのログイン名です。16文字までの英数字(ただし先頭は英字)ですが，8文字までにしておくことをおすすめします。識別名ですから，当然ほかと重なってはいけません。
- 項目2(*)：パスワードが暗号化されて保存されます。この欄はあとでpasswd(8)を利用して設定します。‘*’(アスタリスク)は，ログインが禁止されていることを表します(*2)。パスワードを設定するまではログイン禁止にしておくのが安全でしょう。
- 項目3(1001)：ユーザーに対応するID番号です。1000または1001から順に使っていくとよいでしょう。
- 項目4(20)：ユーザーは，ひとつ以上のグループに属します。グループは，グループ名とグループID番号で表され，この対応関係は，/etc/groupというファイルに格納されています。通常は，一般ユーザーを表すstaff(ID:20)でよいでしょう。ユーザーを複数のグループに属させるには，/etc/groupファイルを利用します。詳しくは，group(5)を参照のこと。
- 項目5(空欄)：NetBSDでは利用されません。
- 項目6(0)：パスワードは定期的に変えることが望ましいわけですが，この項目は，一般にこのパスワード変更を強制するために用いられます。ここではこの機能は利用しませんので‘0’とします。

Unixとテキストエディタ

column

Unixに特徴的な哲学として，すべてがテキスト指向であることが挙げられます。たとえば，各種の設定はすべてテキストファイルとして保存され，またUnixでよく利用されるテキスト整型ツールであるroffやTeXでも，制御コマンドをテキストで記述します。また，表計算をする場合も，データをテキストファイルに保存したうえで，awkやsed，最近ならPerlやRubyといった汎用のテキスト処理ツールで一気に計算するといったことが広く行われます。

こうした事情もあって，Unixで動作するテキストエディタには非常に優秀なものが数多くあります。viのほかに人気のあるエディタといえば，なんといってもGNU Emacs(パッケージeditors/emacs)です。バージョン20では複数の言語を同時に扱う機能(Mule)が統合されました。また，GNU Emacsから派生した，XEmacs(editors/xemacs)も，安定度の向上とともに人気を集めるようになっています。

標準のviでは日本語が扱えませんが，これも複数の言語を扱えるよう拡張したnvi-m17n(editors/nvi-m17n)があります(m17nとは，multilingualizationの略です。mとnの間に17文字省略されていることを表します。余談ですが，これらの多言語化はいずれも主に日本人の仕事です)。

Emacs，viともに，使えば使うほど手に馴染む設計になっています。Unixを使っていくうえでは，いかにテキストエディタを活用するかがキーのひとつとなりますので，どんどん使い込んでやってください。

- ●項目7(0):アカウントが失効する時間を表します。0は,永遠に失効しないことを表します。
- ●項目8(Minoura Makoto):ユーザーのフルネームです。
- ●項目9(/home/minoura):作業用ディスク領域を,ユーザーごとに設定することができます。これを,ホームディレクトリといいます。通常,/home/<ログイン名>です。
- ●項目10(/bin/sh):ユーザーがログインしたときに実行されるシェルというプログラムを指定します。sh(1),ksh(1),csh(1)などが標準で用意されており,それぞれ/bin/sh,/bin/ksh,/bin/cshと指定します。どれでも構いませんが,ここでは/bin/shとしておきましょう。

自分のアカウントを追加し終えたら,エディタを終了します。これで自動的にユーザー情報のデータベースが更新されます。

次に,ホームディレクトリを作成します。

```
# mkdir /home/minoura
# chown minoura:staff /home/minoura
```

Unixでは,すべてのファイルとディレクトリは,あるひとりのユーザー(仮想的なユーザーも含む)とひとつのグループに所有されます。chown(8)は,rootにのみ実行できるコマンドで,あるファイルやディレクトリの所有者/グループを変更するためのコマンドです。ホームディレクトリは,該当ユーザーの所有になります。さて,先ほどパスワードを設定しませんでしたから,すぐにパスワードを設定しておきましょう。

```
# passwd minoura
Changing local password for minoura.
New password:_
```

passwdコマンドは,rootが実行した場合,任意のユーザーのパスワードを変えることができます。それ以外のユーザーは,自分のパスワードのみ変更できます。再度vipwを実行してみると,ユーザーminouraのパスワード欄に,暗号化されたパスワードが記録されているのがわかるでしょう。

以上で,ユーザーminouraの登録が終わりました。

以後,通常の利用時には,この新しく登録したユーザーでログインして作業します。システムの管理操作はrootで行う必要がありますが,このようなときのために,Unixには一時的にユーザーがスーパーユーザーになりかわるためのsu(8)というコマンドが用意されています。通常,管理操作を行う際にはrootでログインするのではなく,ほかの一般ユーザーでログインしたあと,su(8)で一時的にrootに切り替える(これをroot権限を得る,rootになる,などといいます)ようにします。最初にrootでログインしたときのメッセージはこれを表していたわけです。

NetBSDの場合,rootにsu(8)できるユーザーは,wheelという特別なグループに属していなければなりません。グループの設定は,/etc/groupというファイルで行います。

```
# vi /etc/group
```

ファイルgroup(5)も,passwd(5)と似たような構造になっています。ここでは,wheelグループに先ほど登録したminouraを追加してみます。

```
wheel:*:0:root,minoura
```

登録したらエディタを終了し,rootでのログインセッションを終了します。これをログアウトといいます。ログアウトするには,exitというコマンドを投入します。

```
# exit
logout
NetBSD/x68k(myme)(console)
login:_
```

(*2)ログイン禁止:よく見ると,ほかの仮想的なユーザーもログインが禁止されているのがわかります。

Unixのコマンド

登録した新しいアカウントでログインしなおすと,コマンド受け付け状態になります。ここで投入できるコマンドは,Unix一般と同様ですので,市販のUnix入門書を参照することをおすすめします。特に,コマンドのサーチパスの設定は必ず行っておきましょう。rootの環境設定ファイル/root/.profile(シェルが/bin/shまたは/bin/kshの場合)や/root/.cshrc(/bin/csh)が参考になるでしょう。

とりあえずman(1),ls(1),less(1),cp(1),mv(1),sh(1),csh(1)といったオンラインマニュアルを引くところから始めてはいかがでしょうか。

PPPネットワークの設定

次に,PPPを利用してダイアルアップPPP接続をする方法を簡単に説明しましょう。詳しい説明は,オンラインマニュアルpppd(8)などを参照してください。

NetBSDでPPPを利用するには,pppd(8)というコマンドと,ppp(4)というデバイスを用います。これらの設定は,基本的には/etcディレクトリにpppというサブディレクトリを作り,この下に各種の設定ファイルを置くことで行います。

/etcディレクトリの下にファイルを作ることができるのはrootだけですので,rootで作業します。

```
myname% su root
Password:_
```

ここで入力するのは,rootのパスワードです。

```
myname# mkdir /etc/ppp
myname# cd /etc/ppp
myname# mkdir /etc/peers
```

さて,ここに作成するファイル群について説明しましょう。

```
/etc/ppp/options
```

すべてのPPP接続先に対して共通のオプション群を,1行にひとつずつ記述します。

- ●crtscts　ハードフロー制御を行うことを指定します
- ●kdebug 0　デバッグメッセージを抑制します
- ●tty00　内蔵RS-232Cポートを使います。tty00は内蔵RS-232Cポートの名前です
- ●38400　接続スピードです
- ●modem　モデム経由で接続します
- ●defaultroute　デフォルトのネットワークをこのPPP接続に設定します
- ●noipdefault　IPアドレスは相手の指定に従います

```
/etc/ppp/peers/ISP1
```

接続先に応じたオプション群を,やはり1行にひとつずつ記述します。ファイル名は,接続先を表す適当なもの(入力しやすいものがいいでしょう)をつけます。ここではISP1としました。

- ●user minoura　プロバイダのアカウント名を指定します
- ●remotename isp1　接続先名を任意に指定します
- ●connect "chat -f /etc/ppp/chatscript-isp1　モデムの初期化コマンド(後述)のファイル名を

図3　X Window上で動くNaviator



指定します

```
/etc/ppp/chatscript-isp1
```

モデムの初期化コマンドを記述します。接続先の電話番号も含みますので，当然接続先ごとに異なることになります。このファイル名は，/etc/ppp/peers/ISP1ファイルに記述したものです。

以下の例は，NECのAterm55を用いてISDN接続(同期→非同期変換により，同期64kbpsで接続する場合)するために用いているものです。ATコマンドに関しては，モデムなどのマニュアルを参照してください。Windowsなどと違い，残念ながら市販されているあらゆるモデムの初期化コマンドをデータベースに持つなどという力技は使えませんので，個別に調べる必要があります。また，詳しい形式は，chat(8)を参照してください。

- TIMEOUT 3　タイムアウトを3秒に設定します
- ABORT BUSY ABORT 'NO CARRIER'　これらを受け取ると異常終了します
- '' ATZ　モデム初期化
- OK AT$N1=1　"OK"の文字列を受けて，64Kbps同期接続のコマンド
- OK ATD754-XXXX　"OK"の文字列を受けて，ダイアルのコマンド
- TIMEOUT 20　タイムアウトを20秒に変更します
- CONNECT　'CONNECT'を受けて正常終了します

```
/etc/ppp/pap-secrets
```

接続先が，認証方法としてPAP(Password Authentication Protocol)を利用している場合，そのパスワードをこのファイルに記述します。CHAP(Challenge Handshake Authentication Protocol)を用いている場合は，/etc/ppp/chap-secretsを用います。いずれも1行1エントリで，1エントリは，<アカウント名> <接続先> <パスワード> の形式です。

```
minoura isp1 PASSWORD
```

このファイルは，root以外には読み書きできないよう設定しておく必要があります。詳しくは，chmod(1)を参照してください。

```
# chmod 600 /etc/ppp/pap-secrets
/etc/resolv.conf
```

このファイルはPPPとは直接関係なく，英数字からなるホスト名(www.softbank.co.jpのようなと，4つの数字からなるIPアドレス(203.182.230.200のような)の対応付けを行うネームサーバの指定をするためのものです。

ネームサーバはIPアドレスで指定しますが，接続先のプロバイダの指示に従うようにします。

```
nameserver 10.25.192.15
nameserver 10.25.192.1
```

複数のネームサーバが指示されていた場合は，このように2行に分けて書きます。

以上のファイルを用意すれば，PPP接続ができるはずです。suでroot権限を得ている場合，元のユーザーに戻るためにexitと入力します。

```
# exit
myname% _
```

接続には，pppd(8)コマンドを利用します。

```
myname% /usr/sbin/pppd call <接続先>
myname% _
```

<接続先>の部分は，先ほど作成した/etc/ppp/peers/<接続先>のファイル名に利用したものです。モデム(またはTA)のランプで接続を確認してください。また，ifconfig(8)コマンドで，

```
myname% /sbin/ifconfig ppp0
ppp0: flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 1500
inet 10.255.144.187  10.25.144.3  netmask 0xffffff00
myname% _
```

flagsに，'UP'があることを確認してください。この場合，10.255.144.187が自分のIPアドレスになります。

さて，切断には以下のコマンドを用います。

```
myname% kill `cat /var/run/ppp0.pid`
```

複数のコマンドを組み合わせています。このあたりの組み合わせ方が実はUnixの真髄ともいえるわけですが，ここではこれを決まり文句だと思ってください。興味のある方は，Unixの入門書を読むか，sh(1)やkill(1)，ps(1)などを参照してください。

このあとメールの設定などをする必要があるわけですが，ここでは割愛します。

パッケージの利用

NetBSDはUnix互換OSで，Unix向けのさまざまなプログラムが利用できると紹介しました。Unixの世界では，伝統的に多くのフリーソフトウェアがソースコードの形態で配布されており，個別にコンパイルして利用することになります。一口にUnixといっても，さまざまなベンダが微妙に異なる'Unix'を出していることから，それぞれの間の移植性を高めるために，autoconfやimakeといったさまざまな技術が開発されてきています。残念ながら，これらの技術も完璧ではなく，どちらかといえばマイナーなプラットフォームであるNetBSDではソースコードの修正を必要とする場合もあるようです。

NetBSDは，サードパーティ製のソフトウェアを，統一的な手段で簡単にインストールするための仕組みであるパッケージシステムを備えており，数百に及ぶサードパーティ製のソフトウェアを，パッケージとして提供しています。このパッケージシステムは，もともとFreeBSD向けに開発されたものをNetBSDに移植したものです。また，パッケージそのものも，FreeBSDから移植されたものが多いようです。

パッケージには，ソースパッケージ(pkgsrc)とバイナリパッケージの2種類の形態があります。ソースパッケージは，サードパーティから配布されているソフトウェアのアーカイブ(distfile)を取得し，必要であればNetBSD向けに修正をしたあとコンパイル・インストールするまでの手順とパッチファイルなどをまとめたもの，バイナリパッケージはインストールした結果のバイナリファイル群をまとめたものです。先ほどjoeをインストールしたときにpkg_add(8)というコマンドを利用しましたが，これは実はjoeのバイナリパッケージをインストールしたのです。

全パッケージのリストが，

http://www.jp.netbsd.org/ja/JP/Documentation/Packages/list/README.html

からたどれますので参考にしてください。ただし，このリストはNetBSD-currentのものです。

なお，基となったFreeBSDでは，ソースパッケージのことをports，バイナリパッケージのことをpackagesと呼んでいます。

バイナリパッケージの利用

The NetBSD Projectが提供しているバイナリパッケージは，

ftp://ftp.netbsd.org/pub/NetBSD/packages/<version>/<arch>/

にあります。<version>は，対応するNetBSDのバージョン，<arch>はアーキテクチャ(X680x0ならx68k)です。この内容は国内にもミラーされていますので，そちらを利用することをおすすめします。

このインストールは実に簡単で，rootになってから，

```
# pkg_add ftp://www.jp.netbsd.org/pub/NetBSD/packages/1.4.1/x68k/All/ja-FreeWnn-lib-1.10.tgz
```

とURLを指定して入力するだけです。あるいは，あらかじめファイルをftp(1)で取り寄せておき，そのファイル名で指定することも可能です。

```
# ftp ftp://www.jp.netbsd.org/pub/NetBSD/packages/1.4.1/x68k/All/ja-FreeWnn-lib-1.10.tgz
# pkg_add ja-FreeWnn-lib-1.10.tgz
```

インストールされているコマンドは，pkg_info(8)で調べることができます。

```
# pkg_info
joe-2.8:
ja-FreeWnn-lib-1.10:
# pkg_info joe-2.8
```

パッケージのなかには，ほかのパッケージに依存している，つまり，あらかじめほかのパッケージをインストールしておかないと，インストールできないものもあります。たとえば，パッケージja-FreeWnn-server-1.10は，ja-FreeWnn-lib-1.10に依存しています。ja-FreeWnn-lib-1.10をインストールすることなくja-FreeWnn-server-1.10をインストールしようとすると，まずカレントディレクトリにja-FreeWnn-lib-1.10.tgzがないかを調べ，見つかれば先にja-FreeWnn-lib-1.10をインストールします。見つからない場合，その旨表示したうえで異常終了します。

残念ながらx68k(を含む，m68kファミリ)向けにコンパイルされた公式なバイナリパッケージ

は，ソースパッケージに比べてあまり多くはありません。これは，コンパイルに何時間もかかることが多く，リリース時にバイナリまで準備することができないためでしょう。

ソースパッケージの利用

ソースパッケージは，付録CD-ROMのnetbsd/source/pkgsrc.tgzにひとまとめにされています。.tgzというサフィックスは，先ほどバイナリパッケージにも使われていました。Unixでは非常に一般的な圧縮アーカイブ形式ですので，展開方法も覚えておきましょう(*3)。

標準の展開場所は，/usrです(*4)。/usr以下にはroot権限がないと書き込むことができません。

```
# mount -o ro -t cd9660 /dev/cd0a /mnt
# cd /usr
# tar zxvvf /mnt/netbsd/source/pkgsrc.tgz
```

/usr/pkgsrcというディレクトリが作られ，その下に多くのサブディレクトリが作られたことを確認してください。

まず，コンフィギュレーションファイル/etc/mk.confを作成します。これは，サンプルが/usr/pkgsrc/mk/mk.conf.exampleにありますので，これをコピーして雛型とするのがよいでしょう。

```
# cp /usr/pkgsrc/mk/mk.conf.example /etc/mk.
conf
# vi /etc/mk.conf
```

mk.conf(5)は，Makefile形式ですので，'#'文字以降がコメントとなっています。ほとんどの行がコメントになっており，必要な部分の'#'を削除することで設定する形式になっています。

このなかでいちばん重要なのが，ミラーサイトの設定です。GNUソフトウェア，X Window Systemのソフトウェアなど著名なftpサイトは，世界中にその内容のコピーを持っているミラーサイトが存在します。ソースパッケージは，特に指定がなければそれぞれdistfileを一次配布元(マスターサイトと呼ぶ)から取り寄せようとします。これは一般に時間がかかるだけでなく，途中のネットワークにいらざる負荷を掛けてしまい，望ましくありません。そこで，パッケージシステムでは，これら著名なftpサイトに限りミラーサイトの指定をできるようにしています。

日本向けの設定は，mk.conf.exampleのいちばん末尾付近にあります。すべてコメントになっていますので，行頭の'#'をすべて削除してください。

そのほかの設定も，コメント部分を見ればおおむねわかるようになっていると思います。

次に，必要なパッケージの収められているディレクトリを探します。先ほど紹介した一覧のページでもわかるように，各パッケージはarchivers，audio，といったカテゴリに分かれて収められています。

ここでは，日本語対応のless(1)であるjapanese/jlessをインストールしてみることにしましょう。

```
# cd /usr/pkgsrc/japanese/jless
# make install
```

このとき，システムがインターネットに接続されていれば，distfileの取得からコンパイル，インストールまでがこれだけのコマンドで自動的に実行されます。make(1)が正常終了したら，pkg_info(8)でそれを確認してみてください。

(*3).tgz：tar(1)でひとまとめにしたアーカイブを，gzip(1)で圧縮したもので，通常のサフィックスは.tar.gz。CD-ROMやMS-DOSのファイルシステムのように，ファイル名に制限のある場合に.tgzが用いられます。
(*4)標準の展開場所：実際にはどこに置いてもかまいません。

NetBSD/x68kの活用

さて，NetBSDに特有な事項として，ユーザーの追加法，PPP接続の方法，パッケージの利用法を取り上げて説明しました。

NetBSD/x68k本が絶版となってから久しく，この手のまとまった文書を提供するという長年の願いが実現できました。Oh!X的には本来，インストールと設定が終わってからの活用例なんかを紹介するのが本筋なわけなのですが，今回はここで打ち止めとします。

NetBSDは，Human68kやSX-Windowに比べればはるかに高機能・高性能なOSです。月並みな言葉を使えば，生かすも殺すもあなた次第というわけで，X680x0の新しい可能性を存分に楽しんでください。

・ネットワークサーバとして

昨年11月に秋葉原で開かれたオープンソースまつり(日本Unixユーザー会他主催)では，会場内のDHCPサーバをMacintosh SE/30がまかなっていました。もちろん，NetBSDで動作していたものです。このように，メールサーバ，ネームサーバなどのネットワークサーバの仕事にはCPUパワーはさほど必要とされません。24時間365日の稼働に耐えうるNetBSDならではの仕事です。ただし，イーサネットカードは必須でしょうね。

・電子メール端末として

NetBSDでは，Emacs上で動作するものなど，数々の高機能メールクライアントが動作します。テキストベースの電子メールを読むには，X680x0のパワーでも十分でしょう。

・ハードウェア開発プラットフォームとして

NetBSDの移植性の高さを支えるひとつの技術が，デバイスドライバの構造です。マシン依存部と非依存部の分離が徹底的になされており，たとえばX680x0用の付加ハードウェアとしてPCIバスブリッジを開発したとすれば，そのブリッジ用のドライバ，おそらくは100行未満を書くだけで，その先に繋がる各種PCIデバイスの大半が利用可能です。これらPCIデバイスのドライバは，すでにPC/AT互換機やAlphaなど他機種で広く使われた実績のあるものが大半であり，十分信頼できるものです(もっとも，バイトオーダーの違いでうまく動かないものもあるかもしれませんが)。

・オペレーティングシステム実装の学習研究用プラットフォームとして

もともとBSDは大学での研究成果の集合体ともいえるものですし，その後もThe NetBSD Projectによってより洗練されたものに仕上げられています。ソースコードの綺麗さは，LinuxやFreeBSDの遥か上を行くものといわれています。そして，新しい成果が得られた場合には，ぜひともThe NetBSD Projectに還元してあげてください。世界中のNetBSDユーザーによって利用されることになります。

こうした活用法に関しても，機会があれば紹介してみたいと思います。

最後に，NetBSD/x68k開発チームよりひと言。現在，NetBSD/x68k(の機種依存部分)の開発は，わずか2名で行っています。未サポートのまま残されているSASI互換の旧型ハードディスクやOPMといったデバイスのドライバの寄贈や，その他，さまざまなバグ情報などはいつでも大歓迎です。興味とパワーのある方の協力をお待ちしております。

NetBSD-current

column

NetBSDは，1.3.3，1.4などの番号のついた公式リリースのほかに，開発中の最新のソースコードが常に提供されています。これを，NetBSD-currentと呼びます(-currentと略すことがある)。

NetBSDは現在，全世界の百数十名の開発者によりメンテされており，ソースツリー上ではその改良のために大小さまざまなプロジェクトが走っています。メインツリーにはある程度の安定を見てからマージされることになってはいるものの，不安定だったりそもそもコンパイルできなかったりする可能性もあります。

しかし，それを知ったうえでNetBSDの最新の機能を利用し，また開発に参加する楽しみも捨てがたいものです。たとえば，'99年11月現在，バージョン1.5に向けて次世代インターネットプロトコル(IPv6)のサポートをはじめ，ファイルシステムの高速化を図るSOFTDEP，hpcmips(Handheld PC based on MIPSprocessors，MobileGearIIやカシオペアといったWindowsCEマシン)など新しいアーキテクチャのサポートなどが追加されています。

いまどきのアマチュアハード製作入門（メモリボード編）

すず　まきお　Suzu Makio

最近のPCは高速バスやインテリジェント化が進んでいて，ハードウェア工作を行うにはちょっと困りもの。シンプルな接続でハード工作が楽しめるパソコンは貴重です。ここではアマチュアハードウェア製作を通して，ハードウェアの理解を深めていきましょう。まずはメモリボードの製作を行います。040Exelの作者による解説です。

ヤングマガジンUpper'sで連載されていたち○ぽ/刑事[注1]も，とうとう終わってしまい，なんで「ち○ぽ」と「刑事」の間に「/」があるのだろうといまさらになってから悩みつつ，カップヌードルのCM[注2]を見るたびに「こいつなにしとんねん！」と突っ込みを入れてしまう今日この頃，皆さんいかがおすごしでしょうか。

最近のアマチュアハード界はCPLD/FPGAの普及，開発ツールの低価格化により大きな変化を遂げています。アマチュアでも手軽にLSIを設計できるということは我々にさらなる可能性を提示し，飛躍的な開発力の向上を提供しています。現在，アマチュアが中規模以上のデジタル回路を組むときPLDの使用は避けては通れないものといえます。私はちゃんとしたハードウェアの教育を受けたこともなければ[注3]，現場で働いたこともないアマチュアですが，PLDを使い始めてからいろいろな回路が組めるようになり，PLDを使っていなかったときに比べるとかなりスキルアップしたと思います。

しかし，PLDを使ったことがない人にとって，プログラムによって回路を設計するというのは感覚的に捉えにくいものがあり，それがPLDの使用をためらわせる一因になっているようです。

ここではX68000用メモリボードの製作を題材として，CPLDの実際の使い方を解説すると同時に，DRAMの使い方や高周波回路のノイズ対策などを中心に，具体的に解説していこうと思います。読者対象としては，デジタル回路に対する基本的な知識があることを前提としていますが，初心者の方にもできるだけわかるように解説していくつもりですので安心してください。

注1　丘咲賢作さんの超人気漫画。すばらしい感性と美しい絵で楽しませていただきました
注2　執筆時はゴルバチョフ編が放送されています。私は，この2つ前のカールルイス編が大好きです
注3　大学で徐々に習いつつあります。でも実践的じゃないのが問題です

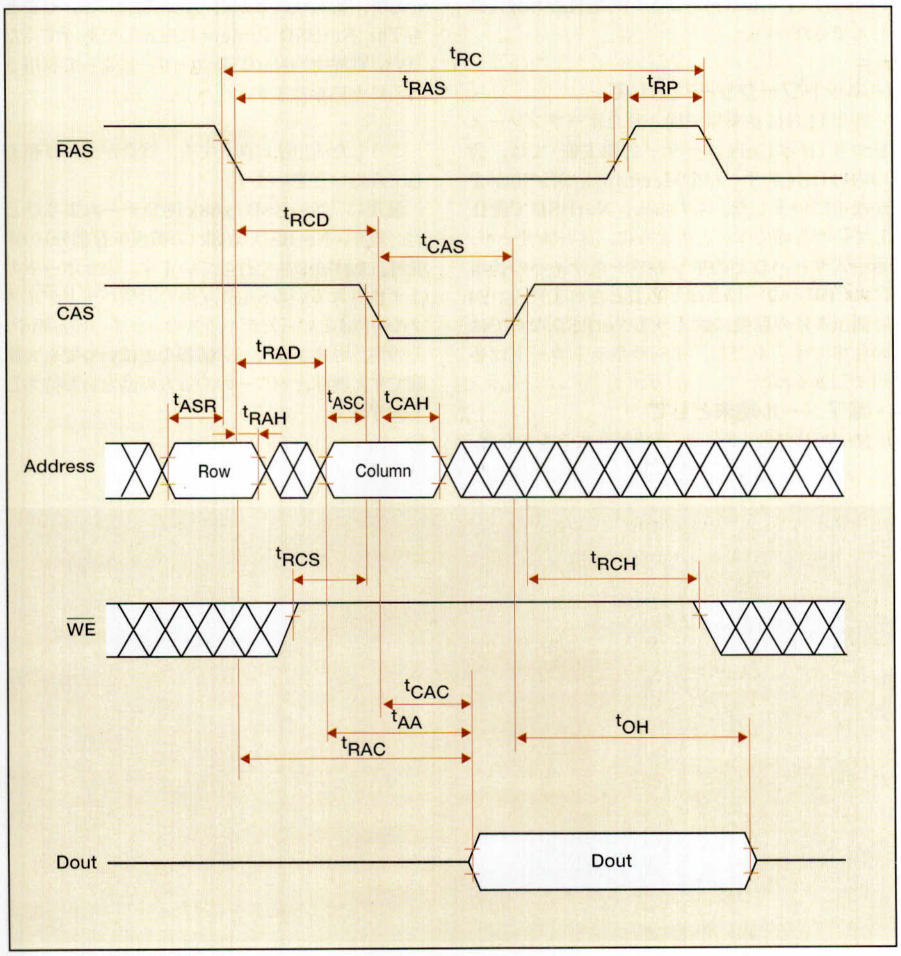


図1　リードサイクル

メモリボードの仕様

まず，今回製作するメモリボードの仕様について解説します。今回製作するのはX68000シリーズ用のメモリボードのうち，拡張スロットを使用するタイプで，市販されている16MBSIMMを利用したものです。16MBのうち10MBを使い，X68000のメモリをフル実装することができます。

DRAMの使い方

今回使うDRAMはFPMのものです。FPMとは，ファストページモードの略ですが，ファストページモード自体は使いません。なぜなら，

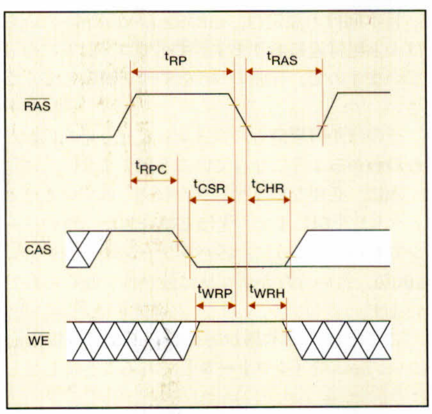


図3　CASビフォアRASリフレッシュサイクル

68000ならノーマルアクセスでもノーウェイトでアクセスできるからです。また，今回の設計ではEDO(注4)のものも使うことができます。

FPM-DRAMは大きく分けて以下の3種類の動作に大別されます。

1）ノーマルリード/ライトアクセス
2）ファストページモードアクセス
3）リフレッシュサイクル

この3つの動作自体も，また細かく内部動作が分かれているのですが，ここでは今回使う代表的な動作についてのみ解説しておきます。

ノーマルリードアクセスのタイミングを図1，ノーマルアーリーライトアクセスのタイミングを図2，CASビフォアRASリフレッシュのタイミングを図3，タイミングパラメータを表1に示します。

DRAMはアドレスがマルチプレクス(注5)されているため，RASの立ち下がりと，CASの立ち下がりそれぞれにRAS（行）アドレス，CAS（列）アドレスを与えなければなりません。DRAMはRAS，CASのネガティブエッジ(注6)でアドレスをラッチし，データを入出力します。リードアクセスとライトアクセスは，W信号で区別し，ほかの信号の動作は，リード，ライトどちらのときでも同じです。リードアクセス時，出力データはtrac, taa, tcac，から規定される時間から，CASがイネーブルのあいだ有効ですが，EDOの場合，RASがイネーブルのあいだ有効になります。それにさえ気をつければ，EDOもFPMと同じように扱うことができます。ライトアクセス時は，CASの立ち下がりでデータをラッチします。

リフレッシュというのは，DRAM特有のもので，これをしないとDRAM内のデータが失われてしまいます。今回はCASビフォアRASリフレッシュというものを使います。このリフレッシュはDRAM内部のリフレッシュカウンタを利用するために，外部回路が簡単になり(注7)，約16 μsごとに1回実行するだけでDRAMをリフレッシュすることができます。このとき気をつけないといけないのは，リフレッシュ時のRAS立ち下がり時はWをHレベルにしておかないといけないということです。私は昔，これに気がつかなかったばかりに正常動作せず，非常に悩んだ記憶があります。

注4　Enhanced Data Outの略。FPMに比べてデータ出力時間が長いので，アクセスタイミングが緩和され，より高速な動作ができます
注5　分割されていること。この場合は，時分割されていることをいいます
注6　クロックの立ち下がりエッジのこと。立ち上がりエッジはポジティブエッジといいます
注7　RASオンリリフレッシュなどでは，リフレッシュアドレスを与えなければなりません

タイミング設計

タイミングパラメータは，すべてを列記すると50項目以上になり，すべてのタイミングを考えると大変なので，必要なものだけを書き出しています。また，この値は一例で，すべてのFPM DRAMがこのタイミングで動いているわけではありません。そのため，十分なマージンをとって

表1　DRAMアクセスのタイミングパラメータ

記号	項目	最小値	最大値
tRC	Random read or write cycle time	104	—
tRP	RAS precharge time	40	—
tRAS	RAS pulse width	60	10000
tCAS	CAS pulse width	10	10000
tRCD	RAS to CAS delay time	14	—
tRAD	RAS to column address delay time	12	—
tASR	Row address setup time	0	—
tRAH	Row address hold time	10	—
tASC	column address setup time	0	—
tCAH	column address hold time	10	—
tRCS	Read command setup time	0	—
tRCH	Read command hold time to CAS	0	—
tCAC	Access time from CAS	—	15
tAA	Access time from address	—	30
tRAC	Access time from RAS	—	60
tOH	Output data hold time	3	—
tWCS	Write command setup time	0	—
tWCH	Write command hold time	10	—
tDS	Data-in setup time	0	—
tDH	Date-in hold time	10	—
tRPC	RAS precharge to CAS hold time	5	—
tCSR	CAS setup time	5	—
tCHR	CAS hold time	10	—
tWRP	WE setup time	0	—
tWRH	WE hold time	10	—

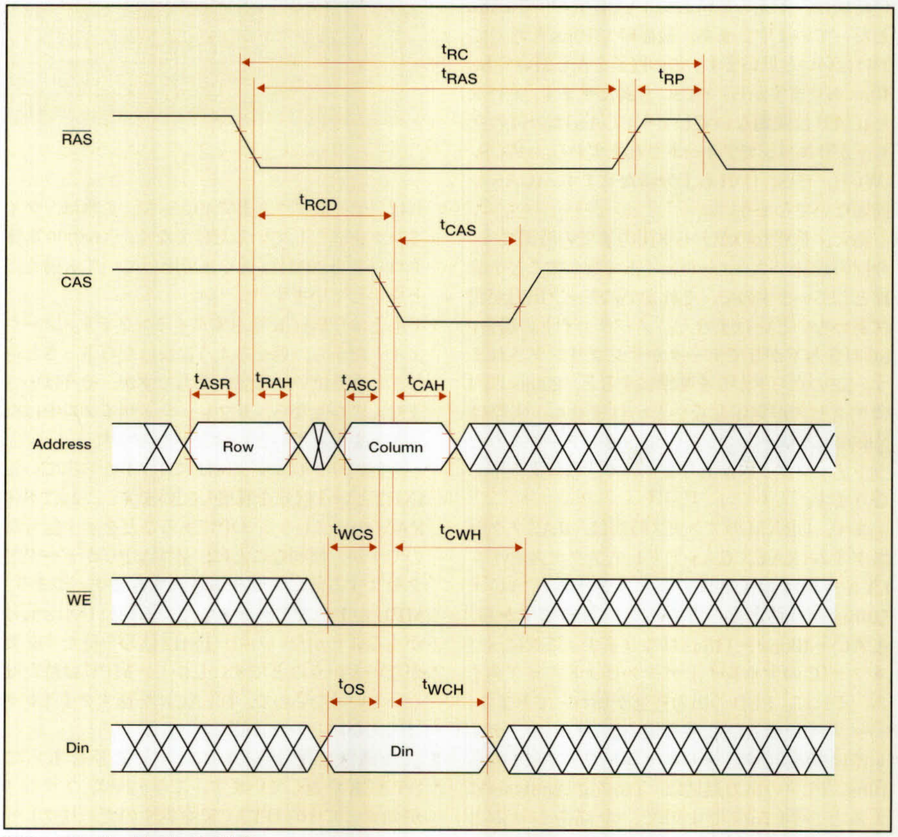


図2　アーリーライトサイクル

タイミング設計する必要があります。

スロットには，68000の信号がほぼそのままきているので，68000のタイミングを基にして設計します。68000のタイミングを図4，タイミングパラメータを表2に示します。

タイミングを考えずに済むような論理を組むと，タイミング設計は楽になります。今回，ASからRASを作り出していますが，そうすると，tRAS，tRP，tRCは，自動的にタイミングは満たせます。また，tRCS，tRCHなどのタイミングも満たせます。

次は，RASからCASとアドレス切り替えのタイミングです。この周りのタイミングは，十分なマージンを取っておくことが安定動作の秘訣となります。なぜなら，この部分はDRAMがRAS信号やCAS信号などで活性化され，電流が一気に流れるために電源やGNDにノイズが発生しやすく，また，アドレス切り替えという多くの信号が一気に変わる動作をするためにクロストークを生じやすくなります。すると信号が揺れるために，ぎりぎりの設計をしていると，タイミングが満たせなくなる場合があります。また，マージンを十分取っておけば，クロックアップにも耐えることができます。

ここでは，RASからアドレス切り替え，アドレス切り替えからCASまでのタイミングを10MHzクロックのデュアルエッジを使って，50nsずつとっています。こうすれば，tRAH，tRAD，tRCDは十分満たせます。今回は，アドレス切り替えにALS157を使っていますが，これのSELECT遅延時間は，手元の規格表でTYPE値[注8]で6.5nsとなっています。まあ，最悪時で10nsと考えておけばいいと思います。この遅延を差し引いても，40nsありますから，tASCも満たせます。tASRとtCAHは問題ないでしょう。CASは，S4の立ち下がりエッジでアサートされますから，tWCS，tWCH，tDS，tDHも十分満たせます。tCASも問題ないでしょう。

あと，重要なのがデータ出力までの時間です。今回の設計では，ノーウェイトで動かすことを前提としていますので，それまでにデータ出力が間にあわないといけません。ノーウェイトの場合，S6の立ち下がりでデータがサンプリングされます。S2の立ち下がりを基準にすると，200ns以内にアクセスが完了しないといけません。実際はX68000本体バッファや，68000のデータ入力セットアップタイムがありますので，これより早くなります。

まず，DRAMのアクセス時間は，RASアクセスタイムtRAC，CASアドレスアクセスタイムtAA＋60ns，CASアクセスタイムtCAC＋100nsのうち，もっとも大きいものですから，tCAC＋100ns＝115nsになります。これに，バッファICのディレイ，データセットアップタイム，それに，PLDの出力遅延が加わってきます。バッファICはALS254を使いましたので，ディレイは10nsです。データセットアップタイムも10ns。PLDの出力遅延は，Timing Analysisによると，14.5nsですので，全部あわせると149.5nsになります。S4の立ち下がりを基準にすれば，データアウトまで49.5nsで，これは，タイミングを満たしているだけでなく，X68000本体クロックを20MHz近くまで上げていても動くことを示しています[注9]。

あとはDTACKを返すタイミングです。ノーウェイトだから，返せるようになったらさっさと返してしまえばいいと思いがちですが，そうはいきません。なぜなら，リフレッシュサイクル中はウェイトを入れないといけないからです。

今回，リフレッシュにはスロットから出ているINH2という信号を利用しています。これは本体RAMがリフレッシュ中であることを示す信号なのですが，便利なことに，必ずMPUがデータアクセスする前にアサートされてきます。つまり，MPUがデータアクセスしている間にINH2信号がくることがないので，INH2信号を待たす必要がないということです。しかし，MPUは待たせないといけないので，DTACKを返すタイミングは考えないといけません。

今回は，アドレスセレクト信号をもってDTACKを返しています。こうすれば，アクセスが始まってからDTACKを返すので，リフレッシュサイクルでは自動的にウェイトが入ります。

リフレッシュサイクルは，スロットから出ているINH2信号と，CASWRUという信号を使います。このCASWRUというのは，ライト動作のときにボード上でCAS信号を作り出すためにあるようですが，実際のアクセスには使いづらいので使っていません。ただ，この信号もCASビフォアRASリフレッシュを考えているようで，INH2がアサート中にも出力されてきます。これを利用してCAS，RASを作れば，リフレッシュサイクルのタイミングも楽に満たせます。

注8　平均値

注9　S4の立ち下がりからS6の立下がりは100nsなので，20MHz動作時は50nsになり，ぎりぎりですが，49.5nsより小さいです。もちろん，ほかの部分もタイミングを満たしていないといけません。今回の設計では，20MHzでも動くように設計しているつもりですが，20MHzでの試験動作はしていません

表2　68000タイミングパラメータ

信号名	記号	項目	8MHz 最小値	8MHz 最大値	10MHz 最小値	10MHz 最大値	12.5MHz 最小値	12.5MHz 最大値
CLK	tcyc	クロック周期	125	250	100	250	80	250
	tCL，tCH	クロック幅L/Hレベル	55	125	45	125	35	125
	tcrf，tcr	クロック立ち下がり/立ち上がり時間		10		10		5
FC_0-FC_2 A_1-A_{23}	tCHFCV	クロックがHレベルからFC有効		70		60		55
	tCHAFI	クロックがHレベルからアドレスFC無効(最小)	0		0		0	
	tCLAV	クロックがLレベルからアドレス有効		70		60		55
$\overline{AS}$	tCHSL	クロックがHレベルから$\overline{AS}$，$\overline{DS}$がLレベル	0	60	0	55	0	55
	tAVSL	アドレス有効から$\overline{AS}$，$\overline{DS}$がLレベル(リード)/$\overline{AS}$がLレベル(ライト)	30		20		0	
	tFCVSL	FC有効から$\overline{AS}$，$\overline{DS}$がLレベル(リード)/$\overline{AS}$がLレベル(ライト)	60		50		40	
	tCLSH	クロックがLレベルから$\overline{AS}$，$\overline{DS}$がHレベル		70		55		50
$\overline{LDS}$/$\overline{UDS}$	tSHAFI	$\overline{AS}$，$\overline{DS}$かHレベルからアドレス/FC無効	30		20		10	
	tSL	$\overline{AS}$，$\overline{DS}$幅がLレベル(リード)/$\overline{AS}$がLレベル(ライト)	240		195		160	
	tDSL	$\overline{DS}$幅がLレベル(ライト)	115		95		80	
	tSH	$\overline{AS}$，$\overline{DS}$幅がHレベル	150		105		65	
R/$\overline{W}$	tSHRH	$\overline{AS}$，$\overline{DS}$がHレベルからR/$\overline{W}$がHレベル(リード)	40		20		10	
	tCHRH	クロックがHレベルからR/$\overline{W}$がHレベル	0	70	0	60	0	60
	tCHRL	クロックがHレベルからR/$\overline{W}$がLレベル(ライト)		70		60		60
	tASRV	$\overline{AS}$がLレベルからR/$\overline{W}$有効(ライト)		20		20		20
	tAVRL	アドレス有効からR/$\overline{W}$がLレベル(ライト)	20		0		0	
	tFCVRL	FC有効からR/$\overline{W}$がLレベル(ライト)	60		50		30	
	tRLSL	R/$\overline{W}$がLレベルから$\overline{DS}$がLレベル(ライト)	80		50		30	
$\overline{DTACK}$	tSHDAH	$\overline{AS}$，$\overline{DS}$がHレベルから$\overline{DTACK}$がHレベル	0	245	0	190	0	150
	tASI	非同期入力セットアップタイム	20		20		20	
D_0-D_{15}	tCLDO	クロックがLレベルからデータ出力有効(ライト)		70		55		55
	tSHDOI	$\overline{AS}$，$\overline{DS}$がHレベルからデータ出力無効(ライト)	30		20		15	
	tDOSL	データ出力有効から$\overline{DS}$がLレベル(ライト)	30		20		15	
	tDISL	データ入力からクロックがLレベル(リードのセットアップタイム)	15		10		10	
	tSHDII	$\overline{AS}$，$\overline{DS}$がHレベルからデータ入力無効(リードのホールドタイム)	0		0		0	

ノイズ対策

DRAMはノイズに非常に敏感なデバイスです。RAMはコンピュータを構成する部品の中でももっとも重要なもののひとつで，ビット化けなどの誤動作は絶対に許されません。完全に安定させる必要があります。そのため，製作には高周波回路に

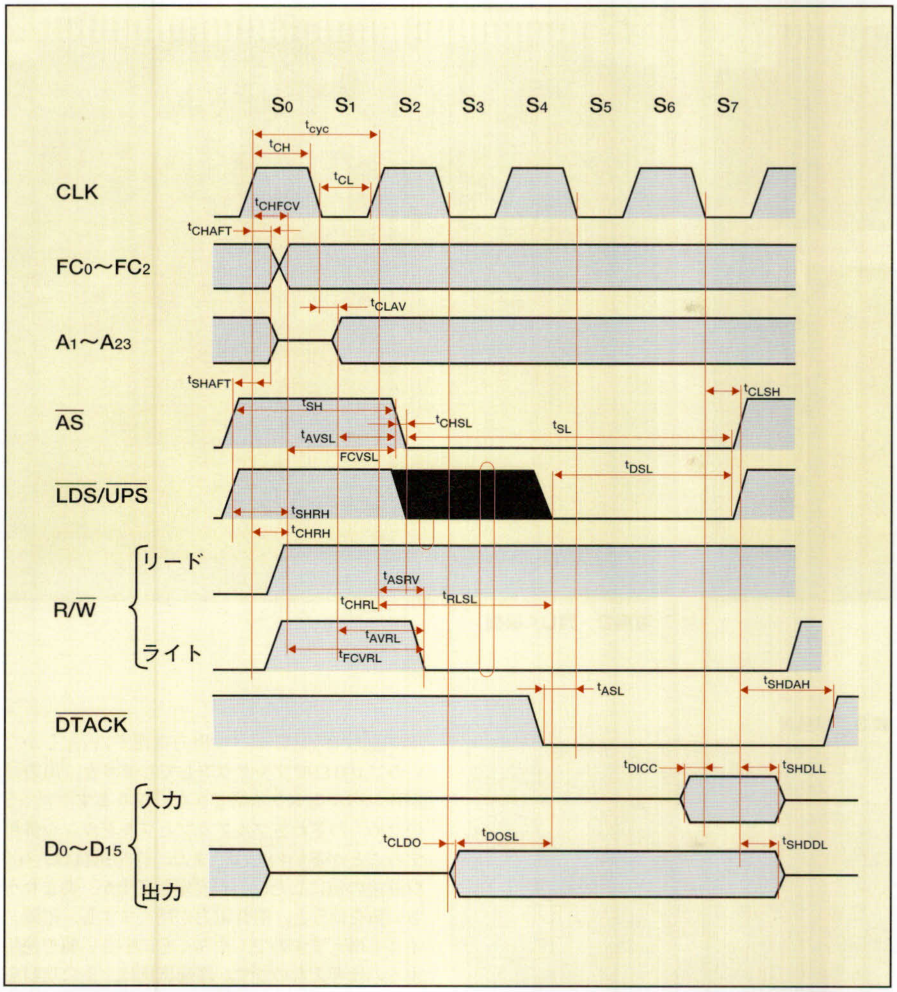


図4　68000タイミング図

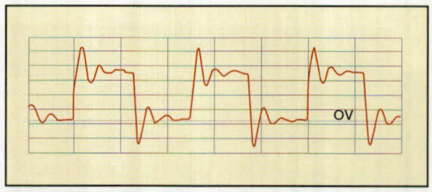


図5　リンギング

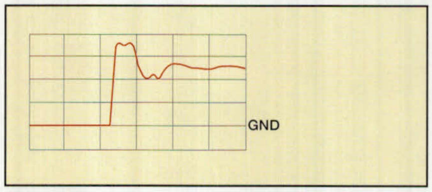


図6　反射

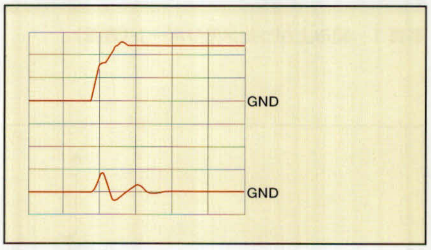


図7　クロストーク

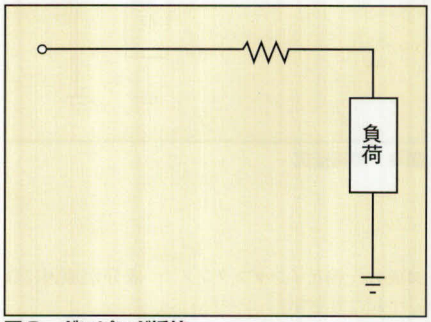


図8　ダンピング抵抗

関する知識や対策が必要になります。高周波回路というのは，単に動作クロックで判断されるものではなく，信号の急峻な立ち上がりにも高周波成分がありますので，ノイズに敏感な回路＝高周波回路，という捉え方をするべきです。

では，高周波回路ではどのような問題が生じるのかということになりますが，代表的なものが，電源，GNDラインのノイズ，リンギングによるオーバーシュート，アンダーシュート，反射，クロストークなどです。これらの現象はすべて，インダクタンス[注10]成分，キャパシタンス[注11]成分，抵抗成分が原因となっています。

配線に電流が流れれば磁束が発生し，磁束が発生すれば電磁誘導が起こるわけですから，配線には必ずインダクタンス成分が存在します。キャパシタンス成分は，配線が1本しかなければ配線自体にキャパシタンス成分は存在しませんが，TTLやCMOSの入力には寄生容量[注12]が存在しますし，実際のプリント基板には多くの配線が存在しますので，これも必ずついてくる問題です。抵抗成分が存在することは簡単におわかりになると思いますが，よっぽど電源ラインを細くしない限り，抵抗成分が問題になることはほとんどありません[注13]し，信号配線は細くしても構いません。

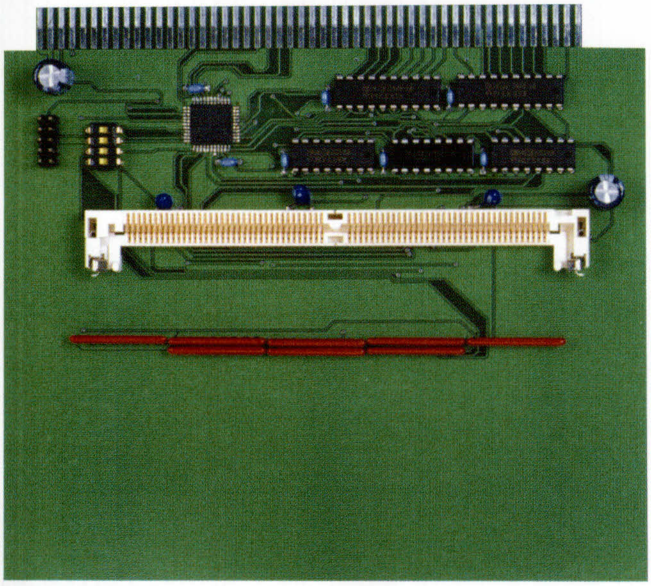
これらの現象が起こるメカニズムを詳しく解説すると少し難しい話になりますので，ここでは簡単に実際の現象と対策を説明しておきます。

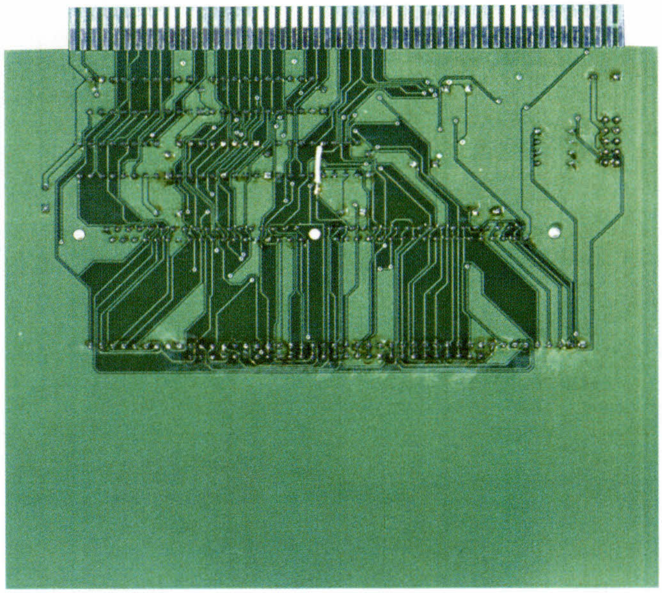
電源，GNDノイズは，デバイスがスイッチング動作を行うときに，一気に電流が流れるために，電源ラインが揺れてしまうために起こります。特に，GNDラインが揺れてしまうと，信号レベルの基準が揺れてしまうことになるので，致命的な結果を招きます[注14]。また，電源ラインのノイズはそのまま信号のノイズになりますし，動作定格を超えると誤動作の原因となります。

リンギングというのは，図5のように，信号レベルがHになったり，Lになったりした直後に，その信号レベルが行きすぎてしまい，またそれが元のレベルに戻ろうとして揺れてしまう現象です。Hレベルになろうとして行きすぎることをオーバーシュート，Lレベルになろうとして行きすぎてしまうことをアンダーシュートといいます。リンギングが発生すると，デバイスの動作入力電圧範囲を超えてしまうために，誤動作を起こす可能性があります。また，信号が揺れるために，一瞬違うレベルにドライブされることもあります。

反射は，図6のようなもので，配線の端で信号が跳ね返ってくる現象です。通常，ドライブした電圧より高い電圧の信号が跳ね返ってきます。これもリンギングと同様，デバイスの動作入力電圧範囲を超えてしまうために，誤動作を起こす可能性があります。また，反射はドライブする側でも起こりますので，このエネルギーが消費されるまで，何度も反射を起こし，周波数の高いクロックでは致命的な結果を招くことがあります。

クロストークは，図7のように，2つの配線の間で起こる現象で，片方の信号のレベルが変わると，もう片方の信号もそれにつれて揺れてしまうというものです。2つの配線の間のキャパシタン


写真１　組み上がったメモリボード(表面)


写真２　同じく裏面

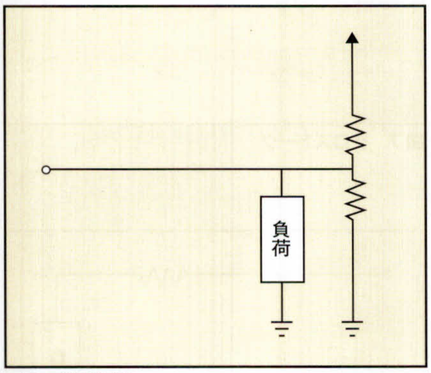


図９　終端抵抗

表３　部品表

M4-32/32-10VC	1
SN74ALS157	3
SN74ALS245	2
電解コンデンサ 470μF	2
タンタルコンデンサ 33μF	3
積層セラミックコンデンサ 0.1μF	7
DIPスイッチ	1
集合抵抗(BI102J924)	6
集合抵抗(BI102J748)	2
SIMMソケット	1

ス成分，相互インダクタンス(注15)成分が原因になって起こります。

電源ラインのノイズはバイパスコンデンサ，通称パスコンで除去することができます。一般的にパスコンとして使われるコンデンサは，低周波ノイズを吸収し，電圧降下を防ぐための電解コンデンサ，中周波ノイズを吸収するためのタンタルコンデンサ，そして，高周波ノイズを吸収するための積層セラミックコンデンサです。前述のとおり，配線には必ずインピーダンス成分が存在するので，高周波ノイズを除去するためのパスコンは，デバイスのすぐ近くに実装し，配線インピーダンスを最小にしなければなりません。一般的に，電解コンデンサは100μ～470μFのものが，タンタルコンデンサは1μ～33μFのものが，積層セラミックコンデンサは0.01μ～0.1μFのものがよく使われます。これらの値は本来，計算によって求めるべきですが，アマチュアならアマチュアらしく，カットアンドトライで値を決めてもいいと思います。

GNDラインのノイズは，回路設計レベルではどうしようもありません。そのため，基板設計レベルで，GNDラインをとにかく太く，短く配線する必要があります。

リンギング，反射は，ダンピング抵抗，終端抵抗で緩和することができます。ダンピング抵抗とは，図8のように，配線に直列に入れる抵抗のことで，10～60Ωくらいのものが使われ，主にリンギングを防ぐ役割をします。終端抵抗とは，図9のように，配線の終端で信号と電源，GNDラインに入れる抵抗のことです。この抵抗は，一見するとただのプルアップ，プルダウンなのですが，通常行われるプルアップ，プルダウンに比べて抵抗値が小さいものが使われます。通常，プルアップ，もしくはプルダウンのみの場合は100Ωまで，プルアップ，プルダウン同時に使用するときは200～300Ωまでのものが使われます。しかし，ここで気をつけないといけないのは出力段のドライブ能力です。Hレベル出力電流が2mAしかないのに100Ωでプルダウンしてしまうと，出力段がHレベルをドライブできなくなります(注16)。そのため，わざわざプルアップとプルダウンを併用することが多いのです。また，終端抵抗は，100Ω前後の値にしたほうが安定しますが，あまり小さい値を使うと，消費電力が増えますし，電源ノイズも増えますので，やみくもに小さい値を使用するのは考えものです。終端抵抗は，主に反射を抑制します。

これらの抵抗の値を正確に計算することは，アマチュアレベルではまず不可能なので，カットアンドトライで試してみるしかありません。

クロストークに関しては，信号の立ち上がりを緩やかにすることによって緩和できます。また，信号の立ち上がりを緩やかにすれば，リンギングや反射も緩和することができます。最近のCPLDは，出力のスルーレート(注17)を操作できるものがほとんどなので，これを利用しますが，標準ロジックデバイスには通常そのような機能はありません。そのため，出力に数p～数十pFのセラミックコンデンサをGNDの間に入れて立ち上がりを遅らせたりすることもあります。

以上，代表的な高周波回路のノイズ対策を述べてきました。このなかでまず必要なのは，パスコンです。これは必ず必要です。ダンピング抵抗，終端抵抗は必ず必要というわけではありませんが，配線が長くなるときや，基板が不安定なときはつけておいたほうがいいと思います。信号の立ち上がりを緩やかにするためのコンデンサはつけないほうがいいです。また，クロストークが深刻な問題になることは少ないようです。

これらは回路設計段階でできる対策がほとんどですが，ノイズ対策はプリント基板設計段階で十分に行っておく必要があります。今回のメモリボ

ード程度の回路規模なら，紙と鉛筆だけでも基板設計できますので，もし，プリント基板をご自分で設計される場合，気をつけておくことは，電源，GNDラインをできるだけ太くすること，配線は極力短くすること[注18]，配線でループを作らないこと[注19]などです。

注10　コイルの成分
注11　コンデンサの成分
注12　ICのピンや，内部配線の静電容量により発生するキャパシタンス成分です。通常，数p～数十pF
注13　抵抗成分が問題にならなくても，インダクタンス成分は深刻な問題になるので，太く短く配線しなければいけません
注14　一般的なLレベル入力最大電圧は0.8V，Lレベル出力最大電圧は0.3Vなので，GNDが0.5V揺れるだけでもうアウトになります
注15　片方で励磁された磁束が，もう片方の配線に電磁誘導を起こすことによって生まれるインダクタンス成分。トランスの原理がその代表例です
注16　一般的なHレベル入力最小電圧は2.0Vですから，必要なドライブ能力は，2.0V÷100Ω＝20mAとなります
注17　LレベルからHレベル，またはその逆への遷移速度
注18　配線が短くなれば，インダクタンス成分が減るなどのメリットがあります
注19　いわゆるループアンテナになり，ノイズを受信してしまいます

回路設計

今回製作するメモリボードの回路図を図10，部品表を表3に示します。これは，基板設計のためのネットリストを作成するために作った図なので，配線がちょっとわかりにくいとは思いますが，ラベルを見ればだいたいわかると思います。

PLDを使用しているために，部品点数が非常に少なくなっています。DRAMのアドレスバス，コントロール信号はノイズに敏感なので，終端してあります。パスコンは，各ICに積セラをひとつずつ，SIMMにタンタルを3つ，そして，基板全体に電解コンデンサを2つ使うつもりで設計しています。SIMMに積セラを使わないのは，メモリがソケット実装のために，基板上に積セラを実装しても，メモリICまでのリードインダクタンスが大きいために，あまり意味をなさないと判断したためです。ちなみに，メモリモジュールの上には，まず例外なく，パスコンが実装されています。SW1～4は，認識させるメモリ容量を操作するためのものです。

また，この回路にはありませんが，アドレスバスにもバッファを入れたほうがいいと思います。

この回路を見て，あれっ，と思われた方がいらっしゃるかもしれません。それは，スロットのアドレスバス，データバスのビットと，DRAMのアドレスバス，データバスのビットが対応していないことについてです。これは，少し考えていただければわかると思います。たとえばデータバスなら，便宜上，D0～D15までの名前をつけていますが，1ビットは1ビットであって，D0をどれにしようが勝手なので，基板設計のときに，配線しやすくするために，こうしてあります。ただし，上位バイトは上位バイト，下位バイトは下位バイトでまとめなければいけません。

実装

基板ができあがったら，いよいよ実装です。今回使用したCPLDは44ピンTQFPパッケージで，ピンピッチは0.8mmです。

CPLDは規模が大きくなると，ピンピッチが狭くなり，実装が難しくなると思われがちですが，BGA[注20]でもない限り，実装はそれほど難しくはありません。TQFPパッケージの実装は，私の場合，とりあえず，ハンダブリッジしてもいいからハンダをしっかりつけ，そのあと，ハンダ吸い取り線で余分なハンダを吸い取って実装します。ハンダ吸い取り線は完全にハンダを除去できるわけではありませんので，うまい具合にハンダが残り，ちゃんと実装することができます。ただ，あまりしつこく吸いすぎると，必要な分まで吸い取ってしまうかもしれませんから，ほどほどに吸い取ります。

そのほか，よく聞く方法では，粘性の低いハンダを使用してハンダをつけたあと，余分なハンダを振り落とすというものもあります。

注20　Ball Grid Arrayの略。パッケージ裏面に格子状にハンダボールと呼ばれるものが並び，基板の穴にはめるかたちで実装するため，基板裏面に端子が現れず，ハンダごてを当てる余地がありません

PLD設計

今回使うCPLDはVantis社のM4-32/32-10VCです。これは，M4シリーズのなかではもっとも規模が小さいものですが，今回の回路には十分なものです。Machデバイスは最近使い出したばかりなので，まだよくわかっていませんが，出力の最大値が3.3Vなので[注21]，3.3Vデバイスに直結できるうえ，Hレベル電圧が低いと，発生するノイズも小さくなるので，私は気に入っています。

開発ツールはDesignDirectCPLDです。これは，DataI/O社のSynarioというCPLD/FPGA開発ツールを原型としてVantisが作ったものです。SynarioはCPLD開発ツールとして，ほぼ業界標準に位置しており，多くのCPLD/FPGA開発ツールがこれに倣っています。DesignDirect CPLDはVantisのHP[注22]からダウンロードできます。また，ISPケーブルの回路図もPDFファイルでアップされています。

Synarioベースの開発ツールでは，ABELというハードウェア記述言語を使うのが普通です。回路図入力もできますが，すでに標準TTLロジックで組まれた回路をCPLDで組み直すといった用途以外では，ABELに比べて入力に手間がかかるため，私はあまり使いません。一から設計する場合は，ABELだけで十分だと思います。

注21　http://www.vantis.com/splash_page.html
注22　最近は他社のデバイスもそういう風になりつつあるようです

ABELの文法

幸い，1999夏号でCPLDの解説記事がありますので，論理の組み方については割愛しますが，ABELの文法については述べられてませんので，ここで述べておこうと思います。

ABELにはさまざまな表記法がありますが，覚えておきたいのは以下のものです。

数値表現

Binary（2進数）	^b
Octal（8進数）	^o
Decimal（10進数）	^d（デフォルト）
Hexadecimal（16進数）	^h

論理演算子・算術演算子・関係演算子（優先順位順）

NOT	!
AND	&
減算	-
加算	＋
OR	#
XOR	$
XNOR	!$
等しい	＝＝
等しくない	!＝
より小さい	＜
以下	＜＝
より大きい	＞
以上	＞＝

割り当て演算子

割り当て	＝
レジスタ割り当て	:＝

ドット拡張子

エッジトリガ	.CLK
出力イネーブル	.OE

宣言

「Module」
モジュール名を宣言します。

「Title」
モジュールのタイトルを宣言します。

「Declarations」
この後にピンやノード，セット，定数の宣言を行います。

「Equations」
このあとに論理記述します。

「End」
モジュールの終わりを宣言します。

「pin」
ピンの名前，ピンナンバーなどを宣言します。

「node」
ノードを宣言します。

その他

コメントアウト	"
セット	[]

これだけ知っていれば，なんとかなると思います。実際の使い方はソースを見ていただくとして，基本的に，C言語のノリで使えばよいでしょう。ABELについて詳しく知りたい方は，大きな本屋さんにそういった関係の本が置いてありますので，そちらを参照してください。

PLDのロジックを組むときに気をつけないといけないことは，同期設計にすること，内部遅延を使わないこと，ハザード[注23]に注意すること，などです。

同期設計というのは，簡単にいえば，レジスタクロックにすべて同じクロックを与えることです。同期設計にすることによって，レジスタの値が同時に変わるため，レジスタの結果をデコードしたときにハザードが生じず，動作状態をつかみやすいのでバグが生じにくいうえに，順序回路[注24]，ステートマシン[注25]などは同期設計でないと設計が大変です。また，多くのCPLDは同期設計にしたほうが消費するロジックが少なくなります。

内部遅延を使ってはいけない理由は，PLDは内部配線の仕方，ロットなどで遅延時間が大きく違ってくるためです。また，ハザードは，PLDに限ったことではありませんが，PLDは高速なため，ハザードによるスパイクノイズが致命的な結果を及ぼす可能性が高いので，注意する必要があ

リスト1

```
MODULE memcon

TITLE 'memoryboard controler'

Declarations

sw1..sw4,inh2,caswru,dtack,r_w,uds,lds,as,a23..a20    pin     43,44,1,2,9,10,11,12,13,14,15,18,19,20,21;
ddir,adsel,g,ras0,cas1,cas2,cas3,cas0,ras2,w    pin   30,31,32,34,35,36,37,40,41,42;
xclk1,xclk2     pin     5,27;

ras,cas,refcas,refras,dtacke    node;

address=[a23..a20];

Equations

!ras := ((address==1) & sw1
       # (address>=2) & (address<=3) & sw2
       # (address>=4) & (address<=7) & sw3
       # (address>=8) & (address<=11) & sw4) & inh2;
ras.clk = !xclk1;

!adsel := !as & !ras & inh2;
adsel.clk = xclk1;

!cas := !adsel & !as;
cas.clk = !xclk1;

!refcas := !inh2 & caswru;
refcas.clk = xclk2;

!refras := !refcas;
refras.clk = xclk2;

!ras0 = !ras & !as
      # !refras;

!ras2 = !ras & !as
      # !refras;

!cas0 = !cas & !uds & !a23
      # !refcas;

!cas1 = !cas & !lds & !a23
      # !refcas;

!cas2 = !cas & !lds & a23
      # !refcas;

!cas3 = !cas & !uds & a23
      # !refcas;

dtacke = ((address>=2) & (address<=3) & sw2
         # (address>=4) & (address<=7) & sw3
         # (address>=8) & (address<=11) & sw4) & !as & !adsel;

!dtack = 1;
dtack.oe = dtacke;

!w = !r_w & inh2;

!g = !ras;

ddir = !r_w;

END
```

ります。ハザードが生じるかもしれない，というときにはとりあえずレジスタ出力にしておけばだいたいOKです。

注23　複数の信号をデコードするときに，信号がずれて切り替わると，一瞬間違ってデコードされて，ひげ状の信号が出力されることがあります。これをハザードといいます
注24　前の状態からいまの状態を作り出す回路を順序回路といいます
注25　カウンタの値によって動作を決める回路です

ロジック解説

リスト1に今回のメモリボードのロジックのソースを示します。

MODULE，TITLEに続いて，Declarationsでピンやノードの記述をします。基本的に，信号名は実際に回路で使われている名前にしたほうがわかりやすいでしょう。sw2～sw4は，実装するメモリの領域を指定するためのものです。ddirはデータバスバッファの方向，adselはアドレスセレクタのセレクト信号，gはデータバスバッファのイネーブル信号，xclk1，xclk2はそれぞれ，10MHz，20MHzのクロックです。

pinの後ろに数字が書いてあるのは，ピンナンバーで，それぞれのピンにロックすることを示しています。また，address＝[a23..a20]の記述はセット割り当て，「..」はレンジ演算子というもので，レンジ演算子は，添え数字に従って2つのbit間を補完するものです。つまり，[a23..a20]は，[a23,a22,a21,a20]を意味しています。また，セット割り当てをすることによって，複数のbit列をひとつの変数として扱うことができ，ロジックの記述が楽になります。

Equationsからロジックの記述に入ります。rasでリード/ライト時のras信号のタイミングを作ります。通常，このように横にAND演算子を並べ，ORを使うときは改行します。すると，横にはすべて1が並ばないと真にならない，縦にはどれか1があると真になる，という風に視覚的に見ることができます。まあ，見やすくすることが肝心なので，(!uds # !lds)のような記述は横に並べても構いません。また，負論理[注26]の信号はこのように反転して使わないと，このような記述は生きてきませんし，そうすることによって，その論理が負論理なのか正論理なのかがすぐわかります。

ここではさっそくセット演算子が出てきます。このようにイコールや，大なり，小なりで真偽を作れるので，記述が非常に楽になり，また，見やすくなります。

この信号は10MHzのクロックの立ち上がりでラッチするので，レジスタ割り当てを使い，ドット拡張子でクロックを！xclk1と指定してやります。反転を使うことによってクロックが反転されるので，ネガティブエッジでラッチすることになります。そして，各記述の最後に；をつけます。ここで，AS信号をデコードしてないのは，ASのタイミングは，S0の立ち下がりから規定されているだけなので，S2の立ち下がりの時点でASがアサートされているとは限らないからです。その代わり，実際に出力されるRAS0，RAS2のところで組みあわせています。

次のadselはrasの半クロックあとにラッチするので，クロックはclk1と指定します。

このような感じで書いていきます。dtackはオープンコレクタ[注27]で出力しないといけないので，.oeで出力をコントロールし，dtack信号を出すときだけローインピーダンス[注28]にするようにします。

最後にENDでくくって終わりです。

注26　信号レベルLのときアクティブである信号。逆はもちろん正論理です
注27　L出力はドライブするが，H出力のときは信号をドライブしない出力。そのため，外部でプルアップする必要があります。IC内部でトランジスタのコレクタがそのまま出力につながっているためこう呼ばれます。CMOSの場合はオープンドレインという
注28　なにかしら出力されている状態。なにも出力されていないとき（ハイインピーダンスのとき）に比べて，電源，またはGNDと，インピーダンス（抵抗）がロー（低い）状態なのでこう呼ばれます

コンパイル，フィッティング

ソースが書けたらコンパイルします。記述が間違っていたらエラーが出ますので，その都度直します。文法エラーがなければコンパイルは成功しますが，まだ安心できません。問題はフィッティング[注29]です。PLDはデバイスによってさまざまな制約があります。ゲート数，配線スペース，レジスタ数，その他，デバイス依存のアーキテクチャによってフィットできないこともあります。とりあえず，ピンフリーでフィットしてみます。フィットできればいいのですが，フィットできなかった場合，より規模の大きいデバイスを使用するなどの対策が必要になります。

フィットできれば，今度はピンロック[注30]してみます。ピンフリーでフィットしたときに割り当てられたピンナンバーでロックすれば，ほぼ最適なピンロックができるようですが，ロジック[注31]に余裕があるなら，基板上での配線を考えたピンロックにしてもいいと思います。ただし，ピンロックの仕方が悪いとフィットできない場合も

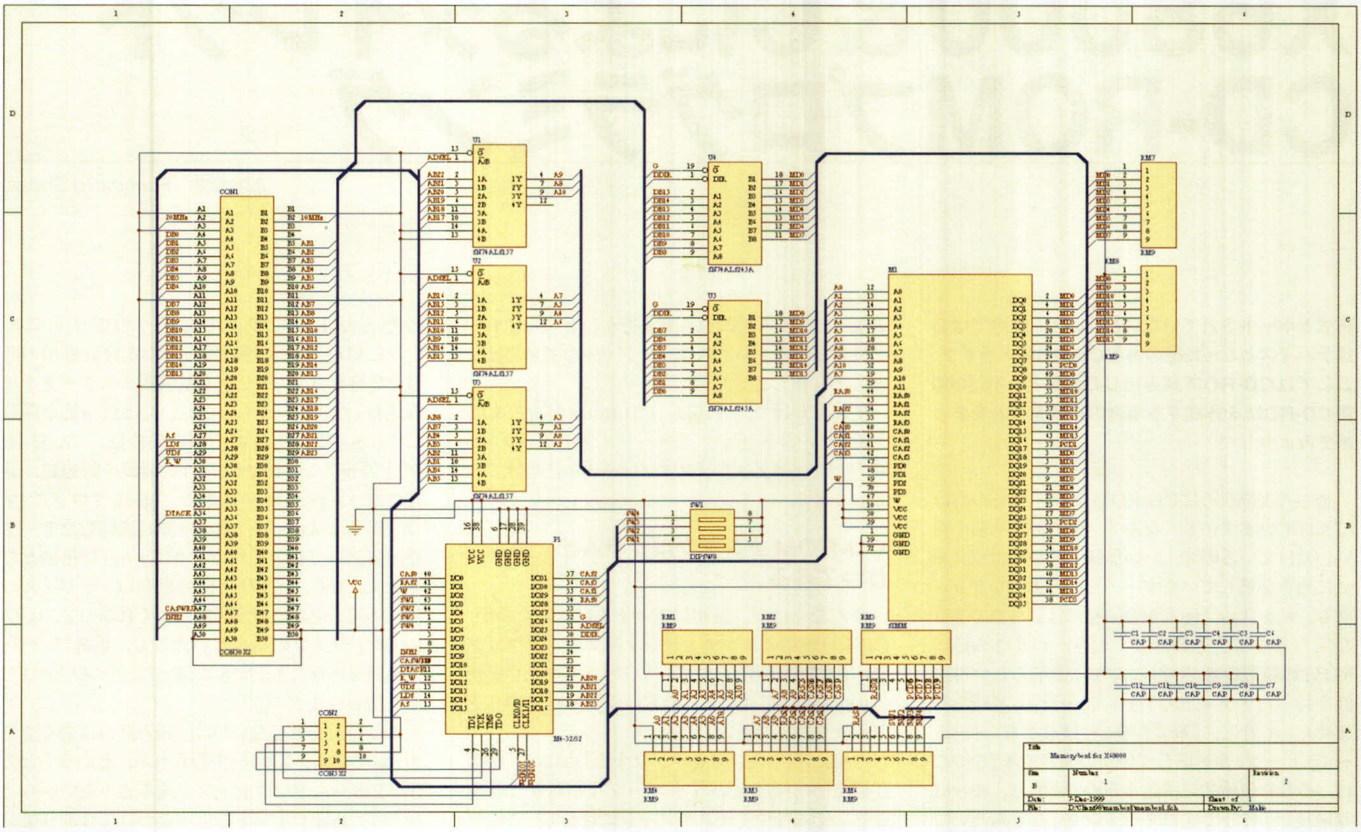


図10　回路図

ありますので，そのときはいろいろ変えて試してみる必要があります。

注29　コンパイルされたロジックを，実際にデバイスに割り当てる作業
注30　デバイスの特定のピンに信号を割り当てること
注31　この場合は配線スペースというべきでしょうか

デバッグ

実装が済んだら，いよいよ書き込み，そして試験動作です。

一発で動くことはほとんどありません。ソフト開発と同様，ハードにもバグはつきものです。アマチュアレベルで，通常，ハード開発でのデバッグで使われるのは，ロジックアナライザとオシロスコープ，もちろんテスタも使います。ロジックアナライザは複数のチャンネルを持っていて，それぞれのチャンネルのある時間の間の信号レベルの変化を見ることができます。複数の信号の状態や変化を一度に見れるので，根本的なロジックのミスや，タイミングが問題の場合には威力を発揮します。オシロスコープは，信号の波形を見ることができますので，ノイズが原因の場合に威力を発揮します。ロジアナやオシロは新品で買うととても高いですが，秋葉などでは中古のものが安く出回っていますので，お金にゆとりがあれば，買ってみてもいいと思います。

私は，今回の製作では，本体をクロックアップしたときにうまく動かなかったのですが，その原因究明にロジアナが非常に役に立ちました。動作不良の原因がわからない場合は，とりあえず，ロジアナなどであたってみるとよいと思います。実際見てみると，予想していた動作とは違う動作をしていることが結構あります。また，CPLDの空きピンをテストポイントとして出しておいて，そこにロジアナのプローブを当てておけば，プローブを付け替えなくても，ロジックを書き換えてテストポイントに調べたい信号を割り当てれば，その信号を見ることができます。こういうところもPLDの便利なところですね。

PLDはデバッグのときに，そのありがたみがいちばんわかります。これまで標準ロジックで組んでいたときは，パターンカットやジャンパなどをしなければいけないので，非常に手間がかかりましたが，PLDなら，ロジックを書き換えてやれば終わりです。ただ，PLDに入っていない信号はジャンパを飛ばしてやらないとどうしようもありませんので，ピン数にゆとりがあるなら，いま使っていなくても，必要になるかもしれない信号は接続しておくことをおすすめします。

だいたい動くようになったら，実際にメモリ上でアプリケーションを動かしてみて，安定動作しているかどうか確認します。デバッガ上で正しく読み書きできても，実際アプリを動かしたときに問題が発生することもあります。

最後に

私の場合，こんな感じでハード開発をしています。今回は触れませんでしたが，基板設計も自分でしています。そもそも，回路設計と基板設計を違う人が担当するのは非効率だと思います。回路設計者は，どの信号がノイズに弱いのかを把握していますので，回路設計者が基板設計もできれば，その信号を優先的に配線することによって，ノイズに強い基板を作ることができます。まあ，そうはいっても，ひとりでなにもかもするのは大変だとは思いますが。

次回はいよいよ，040Excelについての解説記事を予定しています。そこでは，SDRAMの使い方や，今回詳しく解説しなかった高周波回路のノイズのメカニズムと具体的な対策やその計算方法，順序回路やステートマシン記述などの解説をしようと思っています。比較的高度な内容になるので，今回解説したようなことは最低限知っておく必要があります。

それでは，皆さんの開発力向上を願って。

すず　まきお
e-mail:g97b0461@mn.waseda.ac.jp
URL:http://www.geocities.co.jp/SiliconValley-PaloAlto/1700/

X68000&Cによるダイレクト CD-ROMプログラミング

舩本昇竜　Funamoto Shoryu

正式サポートされていないとはいえ，昨今では必須デバイスとなった感のあるCD-ROMドライブ。ここではCD-ROM読み出しの基礎と，X68000用CD-ROMを作成する場合の諸注意などをまとめてみよう。

皆さんX68000にCD-ROMドライブをつなげてあげていますか？　なんて，もう何年も前からいい続けているので，いい加減，呼びかけソノモノに飽きを感じています……というようなジョークも，私を含め，何人かの同胞には許されるような気もします。それはさておき，いまさらCD-ROMの特徴を並べて「こんなに素晴らしい可能性を持ったメディアだ」なんていうつもりはありません。しかし，「利点が多い」けれども「メーカー（シャープ）が対応していないデバイス」をつなげ，それを積極的に活用するのであれば，やっぱり最後は「自分でプログラミング」してみたいですよね。ね！

メジャーだけれどもレアな存在としてのCD-ROM

CD-ROMは，ひとつのメディアに異なった物理フォーマットが平気で複数種類乗り合わせることができる数少ないメディアです。もっともメジャーな例として，CD-ROMでは「データトラック」と「オーディオトラック」が混在し，基本的に，前者はパソコンで，後者はCDプレイヤーで使われているフォーマットです。そして，データトラックに「CDプレイヤーで聞くための音楽」を入れることはできませんし，オーディオトラックに「パソコンで使うためのデータ」を入れることはできないなど，「ルールの違い」も現れてきます。このように，根本的に違う考え方がひとつのメディアに存在するデバイスは，メジャーなモノではほとんど存在しません。

そんなわけで，今回は，CD-ROM特有ともいえる「データトラック」「オーディオトラック」の双方からプログラミングアプローチを仕掛けてみましょう。

CD-ROMダンププログラムを作ってみる

そんなわけで，最初にデッチあげてみたのが，CD-ROM用のデータトラック簡易ダンププログラム，CDDUMP.Cです。XCのライブラリを使うので，LIBCユーザーの方はS_READ.HASをリンクしてください。

さて，「『CD-ROMをダンプする』のには，結構手間がかかる」というイメージってありませんか？　そんなマイナスイメージに対して，このプログラム，基本的な手抜きはもちろんですが，案外，短く収まっています。というのも，いちばん手間がかかるコトが予想されるCD-ROMドライブからのデータ読み込みが，意外にも，

```
S_READ (pos, block, id, size, buf) ;
```

のたった1行でできてしまうのです。このS_READ()という関数は，CD-ROM専用の関数というわけでなく，「SCSI装置からデータを読み込む」ための，どちらかといえば汎用的な関数です。S_READ()の引数と動作は，SCSI-IDがid（0～7：普通は6まで）の装置を対象に，論理ブロックpos（0～）番から，blockブロック（ブロックサイズはsizeで指定）分の連続したデータを読み込み，指定されたバッファbufに格納となります。また，S_READ()関数は，デバイスの種類の違いをある程度吸収してくれるので，CD-ROMドライブの場合といえども，単純に「データを読み込むことができるSCSI装置のひとつ」として扱われます(※)。

ここで，注意したいのは，読み込みはあくまでも「論理ブロック単位」で行われる，ということです。128バイトだけデータを読み込みたいからといって，sizeに128を指定するわけではありません。また，ライブラリのマニュアルを読んでも，1ブロックあたりの容量としてsizeに指定できる値については，

0＝256バイト/ブロック，1=512バイト/ブロック，
2=1024バイト/ブロック

リスト1　S_READ.HAS

```
;  SCSIコール：read
;
;  int   S_READ(pos,block,id,size,buf);
;  int   pos;               読み込み位置（論理ブロック番号）
;  int   block;             読み込み量（論理ブロック数）
;  int   id;                SCSI-ID
;  int   size;              1ブロックの容量（CD-ROMの時は4を指定）
;                           0=256byte,1=512byte,2=1024byte,4=2048byte
;  unsigned char * buf; 読み込み先頭アドレス
;

_S_READ:
   movem.l        d3-d5,-(a7)
   movem.l        $0010(a7),d2-d5/a1
   moveq.l        #$f5,d0                         ;SCSIコール
   moveq.l        #$21,d1                         ;readコマンド
;  moveq.l        #4,d5                           ;CD-ROM専用
   trap           #15
   movem.l        (a7)+,d3-d5
   rts
```

リスト2

```
/*      CD-DA：TOC読み込み
*/

#include        <iocslib.h>  .

/*      →SCSI-2詳細解説 p.251                  0x0324=804      */
unsigned char   toc[804]={ 0x43,0,  0,0,0,0,  2,0x3,0x24,0};
/* SCSIコマンド発行とデータインのバッファを使い回しています */

int     main()
{
        unsigned char   buf[256];       /* 汎用                 */
        int     id=6,i,n;

/*未実装：使えるかどうかチェック         inquiry             */
/*未実装：メディアチェック                testunit            */
        S_SELECT(id);                   /* SCSI-ID              */
        S_CMDOUT(10,toc);               /* コマンド発行         */
        S_DATAIN(804,toc);              /* データIN             */
        S_STSIN(buf);                   /* ステータスインフェーズ*/
        S_MSGIN(buf);                   /* メッセージインフェーズ*/

        printf("TOCデータ長：%02X%02X¥n",toc[0],toc[1]);
        printf("先頭トラック番号：%02X¥n",toc[2]);
        printf("最終トラック番号：%02X¥n",toc[3]);

        for( n=0,i=toc[2];i<=toc[3];n++,i++ ){
                printf("トラック番号：%02x ¥n",toc[4+n*8+2]);
                printf("アブソリュートCD-ROM開始アドレス：%08X¥n",*(int *)(&toc[4+n*8+4]));
        }

}

/* [ EOF ] */
```

としか記述がありません。しかし，CD-ROMは2048バイト/ブロックのデバイス。いったいどうすればいいの？　などと悩んでしまいますが，ここは「size＝4」としてください（隠しパラメータのひとつだと思ってください）。そうそう，CD-ROMの場合「論理ブロック＝セクタ」であることも，キチンと押さえておきましょう。

ここまでわかってしまえば，プログラムの説明は終わったようなモノです。2048バイトのバッファを用意し，CD-ROM（以下の例では，ID=6）の最初から順番に，1セクタ単位で読み込み，表示すれば，もう立派な「CD-ROMダンププログラム」の完成です。

```
unsigned char buf[2048];
S_READ (0, 1, 6, 4, buf) ;
buf内容表示
S_READ (1, 1, 6, 4, buf) ;
buf内容表示
S_READ (2, 1, 6, 4, buf) ;
buf内容表示
        :
        :
S_READ (n, 1, 6, 4, buf) ;
buf内容表示
```

例：順番に「読み/書き」してみる

※：書き込み（CD-RやCD-RW）に関しては，X68000につなげて，HDやMOと同じ書き込み関数を使えば，ウッシッシ……。などと，そんなウマい話はなく，CD-R（ライトワンス）専用のコマンドを使う必要があります。

リスト3

```
S_SELECT       equ        $01
S_CMDOUT       equ        $03
S_DATAIN       equ        $04
S_STSIN        equ        $06
S_MSGIN        equ        $07

SCSI_IOCS      .macro     callname
   moveq.l     #callname,d1
   moveq.l     #$f5,d0
   trap        #15
   .endm

_S_SELECT:
   movem.l     d4,-(a7)
   movem.l     $0008(a7),d4
   SCSI_IOCS   S_SELECT
   movem.l     (a7)+,d4
   rts

_S_CMDOUT:
   movem.l     d3,-(a7)
   movem.l     $0008(a7),d3/a1
   SCSI_IOCS   S_CMDOUT
   movem.l     (a7)+,d3
   rts

_S_DATAIN:
   movem.l     d3,-(a7)
   movem.l     $0008(a7),d3/a1
   SCSI_IOCS   S_DATAIN
   movem.l     (a7)+,d3
   rts

_S_STSIN:
   movem.l     $0004(a7),a1
   SCSI_IOCS   S_STSIN
   rts

_S_MSGIN:
   movem.l     $0004(a7),a1
   SCSI_IOCS   S_MSGIN
   rts
```

SCSIを直接操作する局面

CD-ROMダンププログラムは，S_READ()という非常に便利な関数を使うことで，あまりにもアッサリ完成してしまいました。そもそも，S_READ()はSCSI用関数のなかでも「よく使う手順をひとまとめにし，簡単に使えるようにした」（SCSIコールの）ライブラリであると同時に，「あくまでも『SCSI装置全体でよく使う』モノ」としてまとめられています。

ですから，CD-ROMでしかできないようなコト，たとえば「CD-ROM上のオーディオトラックの演奏」などは，少なくともXCのライブラリレベルでは，まとめられていません。つまり，CD-ROMのオーディオトラックの演奏を実現するには，あるルールに従った手順でSCSIを直接操作し，CD-ROMドライブに「オーディオトラックを演奏しろ」という命令を送る必要があります。

さっそくですが，その手順である「SCSIバスフェーズの推移」の様子を見てみましょう（図1）。一見，ややこしそうに見えますが，基本的に，単純に上から下に向かって，いろんなフェーズが順番に並んでグルグル回っているだけ，とも見えます。つまり，命令の種類はもちろん，例外や条件判定もあるでしょうけれども，順番にフェーズを実行し，名前のとおり，コマンドフェーズで「オーディオトラックを演奏しろ」という命令を発行すればいいようです。

オーディオトラック演奏プログラムを作ってみる

ということで「ルールに従い，CD-ROMにオーディオトラック再生コマンドを発行することで，オーディオトラックを演奏するプログラム」が，ABSPLAY.Cです。このプログラムもXCのライブラリを使うので，LIBCユーザーの方はS_LOW.HASをリンクしてください。

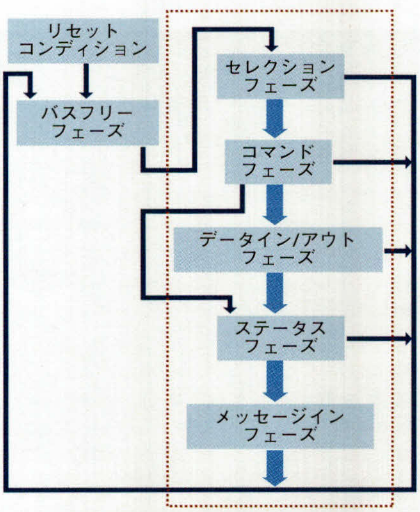


図1　SCSIバスフェーズの推移

このプログラムは，演奏区間のアブソリュートアドレス（絶対アドレスともいう）を16進数で指定することで，オーディオトラックの演奏を行います。アブソリュートアドレスは，CD（-ROM）の先頭を0とし，1秒で75進む，オーディオトラックを扱ううえでの最小単位です。たとえば，普通のオーディオCDをCD-ROMドライブに入れ，

```
@>absplay 2EE 1482
```

とすると，CDの先頭10秒のトコロから1分間，オーディオトラックを演奏します。また，1曲目の演奏時間が3分41秒22，2曲目が2分54秒25，というCDがある場合，

```
@>absplay 40D5 73E8
```

とすることで，2曲目のみ演奏します。絶対アドレスで指定するので，「もっとも細かな演奏時間指定ができるオーディオトラックプレイヤー」ともいえます。

さて，プログラムのほうですが，main()は半ばドーデモイイので，CDDA_absplay()に注目し，注釈，そして図1の「SCSIバスフェーズの推移」とニラメッコしてください。どうですか，セレクション→コマンド→ステータス→メッセージインというように，図1の赤枠内における，データイン/アウトを飛ばす，流れそのものです。それぞれのフェーズの役目を簡単に押さえると，

```
S_SELECT (id) ;
           命令を送りたいSCSI装置をID番号で選択
S_CMDOUT (sizeof (cmd), cmd) ;
           選択したSCSI装置に命令 (cmd) を発行
S_STSIN (buf) ;       コマンドの実行結果を得る
S_MSGIN (buf) ;  インタフェイス制御情報を得る
```

となります。細かく分かれているモノの，1つひとつは比較的簡単に理解できるモノばかりですね。

4つの行それぞれにはちゃんととした意味があり，本来なら戻り値などをキチンと調べなくてはいけません。ですが，ここではもっとも重要なSCSIコマンドの発行部分である，S_CMDOUT()についてのみ解説します。だからといって，

```
S_CMDOUT (sizeof (cmd), cmd) ;
```

の1行だけで，オーディオトラックの演奏ができるわけではないので，くれぐれも，ご注意ください。

SCSIコマンドの発行

SCSIコマンドは，「READ」や「PLAYCDDA」などの人間が理解できる文字列ではなく，nバイトのバイナリデータで表され，SCSI装置の種類にあわせ数多くの種類が存在します。これは本当に多く，実際，1冊の本になってしまったのが，参考文献1です。SCSIプログラミングには欠かせない1冊なので，興味ある人は，傍らに置いて

リスト4

```
/*      CDDUMP
*/

#include        <stdio.h>
#include        <iocslib.h>

int     main(argc,argv)
int     argc;
char    *argv[];
{
        int     pos, pos_st=0, pos_et, len=327680;
        int     block=1, id=6, size=4;          //size:1=512byte/sec,4=2048byte/sec
        int     ac,x,y,ibuf;
        unsigned char   buf[2048];

        for( ac=1; ac<argc; ac++ ){;if( argv[ac][0]=='/' || argv[ac][0]=='-' ){
                switch( argv[ac][1] ){
                        case 'i': case 'I':
                                sscanf(&argv[ac][2],"%d",&id);
                        break;
                        case '$': case 'X':
                                sscanf(&argv[ac][2],"%x,%x",&pos_st,&len);
                        break;
                        case '1': case '2': case '3': case '4': case '5':
                        case '6': case '7': case '8': case '9': case '0':
                                sscanf(&argv[ac][1],"%d,%d",&pos_st,&len);
                        break;
                        default:
                                printf("X68k CD-Dump v0.00 ¥n"
                                "使用法：cddump [/In] [/ブロック,長さ]¥n¥n"
                                "¥tアドレス指定，長さ指定は先頭に x か $ を付けると¥n"
                                "¥t１６進数になります¥n"
                                "¥t/In:nにはCD-ROMのSCSI-IDを設定します(デフォルト=6)¥n"
                                );
                                goto quick_exit;
                        break;
                }
        }}

        pos_et=pos_st+len-1;
        for( pos=pos_st; pos<=pos_et; pos++ ){
                S_READ(pos,block,id,size,buf);
                for( y=0;y<128;y++ ){
                        for( x=0;x<16;x++ ){
                                printf("%02x ",buf[y*16+x]);
                        }
                        for( x=0;x<16;x++ ){
                                printf("%c",(buf[y*16+x]<=0x1f)?'.':buf[y*16+x]);
                        }
                        ibuf=(int)buf[y*16+x];
                        if( (0x80<=ibuf && ibuf<=0x9F) || (0xE0<=ibuf && ibuf<=0xFF) ){
                                printf("%c",'¥x40');
                        }
                        printf("¥n");
                }
        }
quick_exit:
        return(0);
}


/* [ EOF ] */
```

おきましょう。ちなみに3,800円。

さて，ABSPLAY.Cのなかでは，「CD-ROM用コマンド」のなかから「オーディオトラックをアブソリュートアドレス指定で演奏する」というコマンド(PlayAudio(12))を使用しています。プログラムの中では，

```
/* →SCSI-2詳細解説 p.246   start   len */
   unsigned char   cmd[]={ 0xa5, 0,  0, 0,
   0, 0,     0, 0, 0, 0,     0, 0};
```

というように記述しています。この演奏コマンドのデータフォーマットを図2に示します。データフォーマットと照らしあわせてみるとよくわかるのですが，注釈にもあるように，ソース中の，

```
    * (int *) (&cmd[2]) =start;
```

バイト＼ビット	7	6	5	4	3	2	1	0
0	A5h（オペレーションコード）							
1	LUN			Reserved				A※
2	(MSB)							
3	演奏開始							
4	論理ブロックアドレス							
5	(LSB)							
6	(MSB)							
7	転送ブロック長							
8								
9	(LSB)							
10	Reserved							
11	コントロールバイト							

※：A = RelAdr（相対ブッロクアドレッシングフラグ）

図2　PlayAudio(12) コマンド(オーディオトラック再生コマンド)のデータフォーマット

X68000のSCSIって？

column

「X68000のSCSIはかなりマトモだ」とも「かなりヘンだ」ともいわれ，本当のトコロどうなの？　というポジションにいるX68000のSCSIですが，つまるトコロ，「マシン相応のSCSI」てな具合ではないでしょうか？

「マトモだ」派の主張は，X68000のSCSIは「規格としてかなりマトモなSCSI」であるため，「ちゃんと規格に収まったSCSI装置」であれば，たいていつながる。というモノです。これは事実であり，HDやMOを購入するとき，「Macにつながる機種なら大丈夫」というキーワードもそこかしこで聞かれたモノです。と同時に，「××のSCSIは名前だけのオリジナルインタフェイス」と他機種の悪口もよく聞かれたモノです。

さて，「ヘンだ」派の主張は，X68000のSCSIは「SCSIにあるまじき遅さ」であり，一般的なユーザーの不満でもありました。しかし，mach2(p)(※)の登場により，速度に対する不満は，徐々に，聞かなくなってきました。ところが，技術系ユーザーの多いX68000，こんな主張も出てきたのです。「X68000のSCSIには，アービトレーションフェーズがない」。

少々乱暴ない方になりますが，そもそも，アービトレーションフェーズは，イニシエータになることができるSCSIデバイスが同一バス上に複数ある場合における「SCSIバス使用権争奪戦」。しかし，参加者が1名(X68000)で，挑戦者(X68000につなげられる「イニシエータになることができるSCSIデバイス」)が登場する見込みも薄く，無条件で勝者が決まってしまうような状況であるならば，そもそも争奪戦なんて，しなくていいのでは？　というように，「スタンドアロン」かつ「シングルタスク」マシン用のSCSIとして，(当時としても)ごく自然に，アービトレーションフェーズが省略されたコトは，容易に想像することができます。

それでも，やってしまうユーザーがいる，というのが，なんともX68000ユーザーなんですけどね。

※：満開製作所製X68000用SCSI-2カード。バスマスタ方式を採用することで，従来(SHARP SCSI)比で4～5倍の転送速度を実現。

```
*(int *)(&cmd[6])=len;
```

部分において，コマンドの中に開始位置と長さの情報を埋め込んでから，SCSIコマンドを発行していることがわかります。こういった記述を初めてご覧になる方のために簡単に説明しましょう。start，endはともにint型の変数で，4バイトの記憶領域を使用します。上記のような記述により，cmd[2]およびcmd[6]の位置からint型の変数が入るとみなされ，実際にint型の変数を代入することにより，cmd[2]〜cmd[5]およびcmd[6]〜cmd[9]の4バイト（int型の使用領域分）のデータを一度に書き換えることができるのです。

CD-ROMのTOCを読んでみる

最後に，データインフェーズがあるプログラムとして，CDのTOC（Table Of Contents：CDの管理情報）を読み込むプログラム，CDLST.Cを用意しました。このプログラムもXCのライブラリを使います。LIBCユーザーの方は，S_LOW.HASをリンクしてください。

このプログラムは，TOCを読み込み，各トラックのアブソリュートアドレスを表示します。

```
@>cdlst
TOCデータ長：003A
```

「蟲ファイル」の謎を追え

column

さて，CD-ROMといえば，「ISO9660のせいで，8＋3バイトのファイル名」というイメージが強いと思います。とはいっても，実際のトコロ，ISO9660はあくまでCD-ROM論理フォーマットの国際規格であり，名前づけ規則ではないのです。とはいったものの，

種類	ファイル名長さ制限（使用文字制限）
・ISO9660 Level1	8＋3文字（英大文字＋数字＋'_'のみ）
・ISO9660 Level2	27＋3文字（名前のつけ方はISO9660 Level1と同じ）
・DOS準拠	8＋3バイト（S-JIS漢字コードが使用可能）
・WindowsNT3.5準拠	左同

というように，名前づけ規則にも，「ISO9660」の文字が入っているモノもあるので，若干ハナシがややこしくなっています。

X68000の場合，Human68kという限りなくDOSに近いOSを使用しているため，CD-ROMの名前づけ規則は「DOS準拠」を採用すればいいことになっています。「8＋3バイト」のファイル名制限は，ごく局所的に，非常に頭を痛くしてくれますが，まだ漢字などが使えるだけマシかな。というのが正直なトコロです。

ところで，X68000用のCD-ROMを作る場合，現時点ではX68000用CD-Rライティングソフトウェアが存在しないため，よほどのことがない限り，次のような手順を踏みます。もちろん，ファイル名の名前づけ規則は，「DOS準拠」を厳守します。

- MO（IBMフォーマット）をベースに，X68000でコンテンツ製作/動作確認
- Windowsマシンに作業を移し，MOから直接ISO9660イメージを作る
- WindowsマシンでCD-Rを焼き，X68000で動作確認
- マスターのCD-Rを業者に納入

実際，私自身この方法で，数多くのX68000用CD-ROM制作に関わってきました。ところがある日，いつものようにCD-Rを焼こうとすると，ライティングソフトがISO9660イメージ作成中に「ファイル名エラー」を出して止まったのです。そんなハズはない，と思いつつも調査を始めると，どうやら，

```
蟲都002D.DOC
```

というファイルでエラーが発生しているようです（なんてこった，このクサレ「蟲（バグ）ファイル」め，が口からポロリ）。次の瞬間，やはり漢字はだめか？　と一瞬思いましたが，ほかのファイルは問題ありません。「MOがクラッシュしたのか？　それも相当，運の悪いトコロで？」と思いつつ，MOのセクタをディスクエディタでのぞいてみると，ポン！　ちょっと納得。なにを納得したのかというと，MOのディレクトリエントリ部分をのぞいたとき，

```
05 B3 93 73 30 30 32 44 2E 44 4F 43
.ｳ都002D.DOC
```

というデータ列が見つかったのです。少し見覚えがあるのだけれども，どこかが違う。そう，期待したデータは，

```
E5 B3 93 73 30 30 32 44 2E 44 4F 43
蟲都002D.DOC
```

なので，ファイル名の先頭1バイト目のデータが違うのです。

さて，この手のプログラムを組んだことのある人はピンときたと思いますが，Human68kは，ファイルを消去するとき，ディレクトリエントリに記録されているファイル名の先頭1バイトを「E5」にします。つまり，ディレクトリエントリ部分にファイル名のデータがあり，なおかつ先頭1バイトが「E5」以外のとき，Human68kはその名前のファイルが存在すると判断します（ちなみに「00」だとディレクトリの終わりと判断します）。それでは，先頭1バイトが「E5」になってしまうファイル名はどうするのかというと，Humanが「E5」を「05」に書き換えるのです。いわゆる「OS内部での勝手なスリカエ」であり，普段ユーザーが「ファイル名の先頭1バイトがE5だからウンヌン」と気にしなくてもいいようになっているのです。が，気にする局面が発生してしまいました。

先頭1バイトが「E5」の漢字を調べてみると，これが結構，ファイル名の先頭になりそうなモノが多いのです。「薔藪薇藹虱蠅螢褶禅…」これは，マズイかも。それでも，当時は時間的余裕の関係で，問題に該当するファイル名を変更し，リンクが張ってあるなどの，関連箇所を書き換えるというチカラワザで乗り切ったのです。

その後，問題に該当するファイル名がある場合，「暫定的にリネームした状態でISO9660イメージを作り，あとからISO9660イメージを直接書き換える」というワザ（？）を収得し，少なくとも私のX68000環境において，「CD-R上の蟲ファイル」が問題なくアクセスできることを確認しました。さらにその後の，HDの容量に比較的余裕がある，ある日のこと，思い出したように「蟲ファイル」問題で悩まされたMOの内容を「一度，全部HDにコピー」してから，ISO9660イメージを作成してみると，あらら，まったく問題なくできてしまいました。さらには，Windows上から，MOに「蟲.TXT」という名前のファイルを作り，ISO9660イメージを作成してみても，これまた問題ありません。どうやら，私のない頭をさんざん悩ませてくれた「蟲ファイル」問題は，

- MOは「DOSフォーマット済み」のものか，X68000のFIM.Xでフォーマットしたものを使う
- MOへのファイル書き込みはX68000でしか行わない
- ディレクトリエントリ部分に，ファイル名の先頭1バイトが「05」のモノが存在する
- MOから直接ISO9660イメージを作成する

といった条件がすべて揃ったときに現れてくれるようです。ちなみに，CD-Rライティングソフトは，アプリックスのWin-CDRです。ということで，環境が許す人は「蟲」問題を再現させておくと，のちのち安心できると思います。

そうそう，少し前に出てきた，ISO9660イメージを直接書き換えるウンヌンですが，CD-ROMのディレクトリエントリにおいては，ファイル名の先頭1バイトの「E5→05」変換は必要ないようです。つまり「蟲」の場合は，そのまま'E5B3'でよいようです。逆に'05B3'を埋め込むとdirできても，正常にファイルアクセスすることができなくなるので注意が必要です。

先頭バイト	dirで表示され名前	ファイルアクセス
05	_ｳ都002D	不可能
E5	蟲都002D	可能

例：Human＋CD-Rにおける「蟲」ファイルエントリ

```
先頭トラック番号：01
最終トラック番号：07
トラック番号：02
アブソリュートCD-ROM開始アドレス：000080BD
トラック番号：03
アブソリュートCD-ROM開始アドレス：0000B20D
トラック番号：04
アブソリュートCD-ROM開始アドレス：0000D002
トラック番号：05
アブソリュートCD-ROM開始アドレス：0000DE13
トラック番号：06
アブソリュートCD-ROM開始アドレス：0000EE23
トラック番号：07
アブソリュートCD-ROM開始アドレス：0000FAE4
トラック番号：aa
アブソリュートCD-ROM開始アドレス：00010FD4
例：「電脳倶楽部第140号」の管理情報
```

さて，このプログラムは「データインフェーズ」を実行することで，「CD-ROMからデータを受け取る」という点が，オーディオトラック演奏プログラムと大きく異なります。プログラム内では，

```
S_CMDOUT (10, toc) ; /* コマンド発行 */
S_DATAIN (804, toc) ; /* データIN    */
```

の箇所ですね。バッファを使い回していますが，まず，オーディオトラック演奏のときと同じように，CD-ROMに発行する命令「TOC情報を読み出せ」が格納されているバイト列（図3）をS_CMDOUT()に渡します。ちなみに，S_CMDOUT()の第1パラメータは渡すデータサイズ（バイト数）です。

```
/*→SCSI-2詳細解説 p.251  0x0324=804 */
unsigned char toc[804]={ 0x43, 0, 0, 0,
0, 0,   2, 0x3, 0x24, 0};
CD-ROMに発行する「TOC情報を読み出せ」命令部分
```

そのあと，メッセージインフェーズ，S_DATAIN()において「読み出したTOCのデータを配列tocに書き出せ」と指示します。ちなみに，S_CMDOUT()の第1パラメータは受け取るデータのサイズ（バイト数）です。このとき書き出されるTOCデータのフォーマットは図4に示すものになります。

気づかれた方もあるとは思いますが，ここでなにげにSCSIプログラミングのキモが表れています。それは，「コマンドアウトフェーズでは，読み出しデータの位置を指定しない」ということ。つまり，読み込み命令の発行と，実際の読み込みが別々に行われるということです。具体的にいえば，

```
TOC情報を読み出し，〜〜に書き出せ
```

という命令ではなく，

```
CO：TOC情報を読み出せ
DI：TOC情報を〜〜に書き出せ
```

と命令が分離しているということです。些細なコトですが，気づきましたか？

痒いところになんとやら

ちょっと駆け足気味になりましたが，いかがなものでしょうか？

インタラクティブなマルチメディアがナウなヤングにバカうけらしいこの世の中ですが，たまには，こんなふうにプログラムリストに目を向けるのもよろしいかと。優秀なフリーソフトがあれば，ハッキリいって，用のないプログラムたちですが，どのように活用するかは，皆さん次第です。

参考文献

1）SCSI-2詳細解説　菅谷誠一　　CQ出版社

バイト＼ビット	7	6	5	4	3	2	1	0
0	43h（オペレーションコード）							
1	LUN			Reserved			M	R※
2 3 4 5	Reserved Reserved Reserved Reserved							
6	開始トラック							
7 8	（MSB）　アロケーション長 （LSB）							
9	コントロールバイト							

※R：Reserved
M：0＝論理ブロックアドレス形式
1＝MSF形式（分/秒/フレーム）

図3　Read TOCコマンド

バイト＼ビット	7	6	5	4	3	2	1	0
	ヘッダ							
0	（MSB）　TOCデータ長							
1	（LSB）							
2	先頭トラック番号							
3	最終トラック番号							
TOC トラック・ディスクプリタ								
0	Reserved							
1	ADR				コントロール			
2	トラック番号							
3	Reserved							
4 5 6 7	（MSB） アブソリュート CD-ROMアドレス （LSB）							

図4　TOCデータフォーマット

リスト5

```
/*      CD-DA直接再生：アブソリュートアドレスで指定
*/

#include        <iocslib.h>

void    CDDA_absplay(id,start,len)
int     id;                             /* CD-ROMドライブのSCSI-ID      */
int     start,len;                      /* 演奏開始位置／長さ          */
{
/*       →SCSI-2詳細解説 p.246         start       len                 */
        unsigned char   cmd[]={ 0xa5,0, 0,0,0,0,    0,0,0,0,    0,0};
        unsigned char   buf[256];       /* 汎用                          */

        *(int *)(&cmd[2])=start;
        *(int *)(&cmd[6])=len;

        S_SELECT(id);                   /* セレクションフェーズ：SCSI-IDを指定  */
        S_CMDOUT(sizeof(cmd),cmd);      /* コマンドフェーズ：再生コマンドを発行 */
        S_STSIN(buf);                   /* ステータスインフェーズ        */
        S_MSGIN(buf);                   /* メッセージインフェーズ        */
}

void    main(argc,argv)
int     argc;
char    *argv[];
{
        int     id=6,start,end,len;

        if( argc==3 ){
                sscanf(argv[1],"%x",&start);    /* パラメータは16進数で */
                sscanf(argv[2],"%x",&end);
                len=end-start+1;
                CDDA_absplay(id,start,len);
        }
        else{
                printf("X68k ABSPLAY.X¥n¥t@>ABSPLAY/start-addr end-addr¥n"
                "※アブソリュートアドレスを１６進数で指定¥n");
        }
}

/* [ EOF ] */
```

第2回 フェスタ・68レポート column

「X68000だけのイベントを！」という呼びかけから始まったフェスタ・68。1999年10月11日に行われた第2回フェスタ・68の模様と2000年5月4日に開催予定の第3回のご案内などを紹介します。

(しゃかんきょりたもつ)

皆さん第2回「フェスタ・68」へ足を運んでいただきありがとうございました。第2回は当日券も十分用意しておきましたので，動員数が正確に把握できました。しかし連休最後の日に開催というのが無理があったのか，386人といまひとつ伸び悩みましたが，逆にいえば日程に無理があったのによくこれだけの方がご来場くださったといえますね。前売り券は77枚売れてましたから，トータルの参加者は463人で参加サークルさんを含めますと，551人のX68000ユーザーが集結したことになります。

今回の目玉は某社のSCSIボードを，大量にゲットされたサークルさんの叩き売りですねぇ～恒例になりつつある不要030販売とあいまって，異様な熱気を感じましたが，午前中には確かにあった山が午後に入った頃にはすっかりなくなってましたねぇ～。まだまだこの辺りの需要は，あるようですねぇ。

新作ハードのほうは，まきおさん(144ページ参照)の040Excelが，ついに体感デモを行ってました。大勢の方が見いってましたが，予約のほうは順調に取れたのかちょっと心配ですねぇ。急がしいなかサークル参加してくださったまさちくさんは，たくさんの方に囲まれ質問ぜめにあってたようですが，有意義にすごせたのかなぁ～。

ほかのサークルさんも，参加に当たっての苦労にみあうだけのなにかをゲットできたのか心配です。この辺りを気にしだしたら止まらないのでこの辺にして今後の「フェスタ・68」について少し考えてみたいと思います。

今後の課題

今回から「フェスタ・68」ノートを設置しまして，一般参加者の声を足跡というかたちで，いろいろ残してもらいました。目についたのはX68000の初歩的な質問などでしたが，これらに答えるちゃんとした場が現在のX68000界にはないですよねぇ～，ということは新ユーザーが育たないってことになり，高齢化が順調に進んでいき最終的にはX68000の文化がすたれてしまいます。そこで次回からサークル参加してくださる方々に，お願いです！　どんな初歩的なことでも高度なことでも構いませんから，解析/解説本を(CD-ROMでも可)ぜひぜひご出展くださいませ！

初歩的なことは……MOやCD-ROMの繋ぎ方などSCSI周りのことや，OSの設定等々。ちょっとレベルの高い話なら，簡単なソフトの改造法やX68000の仕組みなどをできるだけわかりやすいかたちで，解説していただけると非常に助かります。

個人的には，ぱにっくデータの作り方や，SFXVIのキャラデータの作り方グラフィック，ADPCM，FM音源のいじり方などの，サンプルソフトつきでわかりやすくて詳しい解説本があるとうれしいです。あ，SX-WINDOWの環境構築サンプルっていうのもいいですね。

と，まぁこんな感じで，「フェスタ・68は，新しいユーザー獲得も担っていけるような環境を目指して，可能な限り情報なども含めたいろいろなものを気軽に交換できる場に，皆で育てていきましょう！

そして，「フェスタ・68」で学んだ方が，サークル参加してくださるようになって初めて大成功といえるのかもしれませんね。

次回「フェスタ・68」は2000年5月4日に開催します。今度こそ開催日にサークルカタログを制作したいので，掲載希望のサークルさんは2000年3月15日(必着)までに，参加申し込みをお願いします。それ以降の参加申し込みは，カタログには載せられなくなりますのでご注意ください(Webでの紹介のみとなります)。ただしサークル参加受け付けは，2000年4月15日まで行っています。

以上のことを踏まえてご参加くださるサークルさんは，コラムに記載された項目をハガキなどに記入のうえご郵送ください。

また，採用になるかどうかわかりませんが，サークルカットも募集します。色はスミなどの黒一色で，サイズはB5の4分の1以下でお願いします。

毎回参加してくださってるサークルさん，一般参加の方々どうもありがとうございます。運営を手伝ってくださってるスタッフの皆さんも，前回はお疲れさまでした。

早いもので，「フェスタ・68」も次回でちょうど1周年になりますから，今度もがんばって皆さんと前回以上に盛り上げようと思います。前回と前々回に懲りた方も，少しずつではありますがよい方向へ参加者全員で一丸となって修正に取り組んでいますから，ぜひ！　ぜひ！　ご参加いただいて，大いに盛り上がっていきましょう。

入場前売り券は現在全国のチケットぴあ取り扱い店で絶賛発売中です。購入の際に必要なPコードは<500-130>になります。それでは第3回「フェスタ・68」でお会いしましょう！

2000年5月4日第3回「フェスタ・68」参加サークル募集要項

サークル参加費は1スペース(3000円)，1サークルにつき2スペースまで(6000円)とさせていただきますので，あらかじめご了承ください。

なお，参加費は完全あと払い制ですので，開催当日の9時～10時の間に受け付けまでお支払いください。また当日参加できないときのキャンセル料は，一切かかりません。1スペースにチケット2枚つきです。1スペースのサイズは，180cmの長机半分の45×90cmです。日程はサークル入・退場が9時・16時です。開催時間は10時～15時となっています。

参加希望のサークルは，参加申し込み書(以下の内容を明記したもの)を用意したうえで，下記アドレスまで郵送してください。

ふりがな：
代表者氏名：
郵便番号：
代表者住所：
電話番号：
HP URL：
リンクの可否：(y/n)
ふりがな：
サークル名：
出展内容：(HPとカタログに紹介しますので，できるだけ詳しくお願いします)
X68000の持ち込み：(y/n)
車での搬入：(y/n)

宛先
〒271-0073
千葉県松戸市小根本20-13ワタナベコーポ201号　井上方「フェスタ・68」準備会係

iモード用CGIを作る

大和 哲　Yamato Satoshi

巷で人気のiモード用Webページ。でも，普通のページと２種類のページを作らなくてはならないのはちょっと面倒です。かといってiモードと共用にすると，味気ない……そんな悩みの方のために共用ページの作り方を解説します。

携帯機器用ページの言語たち

最近は携帯電話や携帯機器で「インターネットが使える（Webページが見られる）」ことを謳っているものがとても多くなりましたね。たとえば，今回，私が買ったのはNTT ドコモの「F502i」というiモード携帯電話なんですが，８×８文字分のカラー液晶画面を持っていて，GIFアニメーションなど使ったインターネット上のコンテンツを見ることができます。しかし残念ながら，iモードにせよ，そうでないにせよ，特に携帯電話を使ったタイプ端末では，現状ではすべてのコンテンツが作者の意図したように見ることができるわけではないのですね。ま，手のひらに載るような小さな機械ですから，搭載できるメモリは少ないですし，しかたのないこととはいえ，Webコンテンツ側にもさまざまな制約があるのです。ですので，一般的には，これらの携帯端末で見るためのページを作るにはモバイルアクセス用の仕様の言語を使ったコンテンツを作ることになるわけですね。

ざっと，簡単な基礎知識をお話ししておくと，現在，携帯電話の事業者ごとに携帯機器でのページデータの作り方も3つの流儀があります。それぞれ，HTMLを基本にはしているものの，ホームページを作るための言語として，別々の3つの言語が主に使われています。

iモード用HTML
MML対応HTML
HDML

「iモード用HTML」がNTT ドコモのiモード携帯電話で見ることのできるHTML仕様です。これは簡単にいうとHTMLのサブセットです。「MML対応HTML」はJ-PhoneがJスカイWebで使用しているMML（Mobile Markup Language）という言語に変換するためのHTMLのサブセット。「HDML」はIDO，DDIのEZWebなどWAPサービス対応の言語で，少ないメモリや電話の表示領域にあわせて「デッキ」「カード」といった新しい概念が導入されており，これは従来のHTMLとは少し言語仕様も異なります。

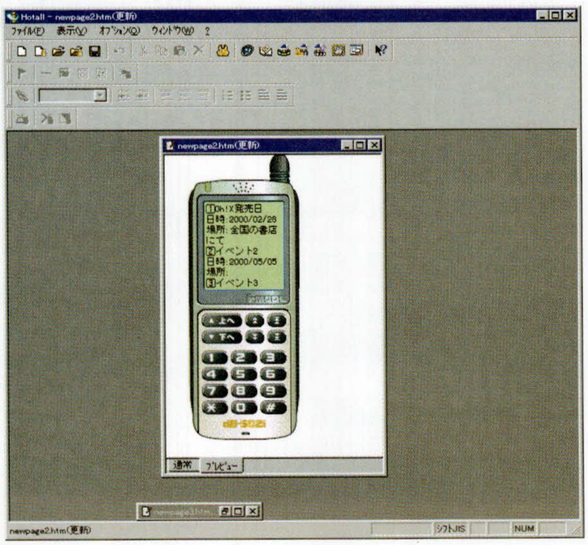


図1　iモード用ページエディタiホタルパーソナル

それぞれの言語に一長一短がありますが，まず，すでにWebページを作ったことのある人にとって，いちばん簡単にデータを作って公開できるのはどれか，ということになるとiモード用HTMLということになるでしょう。というのもHDMLは，通常Webサーバソフト（たとえばApache）のデフォルトのファイルアトリビュートにはまず含まれない，つまり，サーバ側になんらかの設定が必要になってしまうのに対して，「iモード用HTML」は，そのまま「HTML」ですから，サーバの設定も知識もほぼ今までのものがそのまま使えるからです。うまく作りさえすれば，まったく同じHTMLコンテンツで，PC上のブラウザ，あるいはiモード携帯電話，どちらで見てもそれほど違和感のないコンテンツさえできてしまいます。

そこで，今回はサンプルプログラムとして，「iモードでも使えるコンテンツ」作りを目指してみましょう。そして，サンプルとして，

iモード，PCのブラウザどちらでもそれなりに見えるWebページ
iモード，PCのブラウザどちらでも使えるCGIプログラム

を作ってみます。

まずはiモードの基礎

さて，まずは単純にiモード用のHTMLコンテンツを作るにはどうしたらいいかを見ていきましょう（なお，iモード用のページを作る，というだけが目的であったら，たとえばiホタルパーソナルなど，iモード用HTMLに対応したホームページエディタがあるのでそれを使うのが便利です。この手のソフトを使うと，どのタグがiモードで使えるのか悩まずに済みますし，プレビューもあるのでわざわざサーバに転送したりしなくても携帯での表示結果をある程度知ることもできます。極端な話をすればiモード端末がなくても，作ったページの雰囲気くらいはわかってしまいます。市販ソフトって最近は本当にいたれりつくせりですね）。

iモード用のHTMLコンテンツは以下のスペックで作られます。

・iモード対応HTMLを使用する
・文字コードはシフトJISを使用。半角カナと絵文字を使うことができる
・JavaScriptなどは利用できない（CGIやSSIは可）
・コンテンツはインターネット上のWebサーバに置かれる
・サイズは5KB以下。2KB以下推奨
・画像は501iではモノクロ2値。502iでは256色カラー、アニメーション可（94×72ドットが最大）

iモード対応HTML，といってもそれほど特殊なHTMLではありません。iモード用HTMLの仕様はNTT ドコモのホームページ，

iモード対応HTML Version1.0/2.0 タグ・画面イメージ対応表
http://www.nttdocomo.co.jp/mc-user/i/tag/index.html

中に仕様が書かれていますが，基本的に普通にWebページを作るのに使われているHTMLのなかからiモードで表示できる範囲のものを抜粋したものです。このiモード対応HTMLはHTMLの文法がHTML2.0 3.0 4.0のサブセットである，ということ以外は基本的に通常のWebページを作る場合と同じで，Webサーバから見ても同じHTMLコンテンツに見えて区別がつきません。つまり，普通にWebサーバとして動作しているサーバがあれば，そこにiモード対応HTMLで書かれたコンテンツをアップロードしておけば，iモード用ホームページのできあがりというわけです。

サイズに関しては，iモードはとてもシビアです。なぜなら，ブラウザのバッファ領域が5KB，10KBという単位でしかないため，読み込むことので

きるファイルのサイズも「1ファイル5KBまで」と決まってしまいます。多くのiモード対応電話機で，表示できるコンテンツは5KBまでとなっていますので，iモード用HTMLコンテンツを作る場合には必ずそのサイズ内でコンテンツを収めなくてはなりません。ちなみに，501シリーズでは白黒2値の，502の場合は256色カラーのGIF画像をコンテンツに含めることができますが，それを表示するには，画像のサイズも含めて5KB以内に抑えなくてはなりません。

なお，iモード端末のブラウザの実装には3系統あって，

- F501i，N501i
- P501i
- F502i

と分けることができるようです。それぞれで表示できるものなどに多少差異があります。F501iとN501iがジェネリックなiモード対応HTML 1.0用のブラウザ。F502iがiモード対応HTML 2.0用のブラウザです。BASIC認証されたページをアクセスする際に，F501i，N501iでは「同じBASIC認証の領域内でもページを移動するたびにID，パスワードの確認を行う」，F502iでは「同じBASIC認証の領域内であれば一度ID，パスワードの確認をすればiモードを終了するまでは確認なしで移動可能」など挙動に一部違いがあるものの，基本的にはほぼ同じ挙動を示すはずです。また，P501iは，実装が違うようでいくつかのタグでF501iやN501iと違う挙動を示すことがあるそうです。

ですので，以下「DoCoMoの"iモード対応HTML Version1.0・2.0 タグ・画面イメージ対応表"にはこのように書かれているが実機ではこのような動作…」のように書かれているものはP501iでは違う動作をすることがあります（筆者もこの機種については実機を持っていないので確認できていません。なお，このページに記載されている事柄はF502i，F501iのみで確認しています）。基本的な部分，たとえば5KBまでデータを読めるなどの仕様は同じなようですが……。

iモード対応HTMLの言語仕様

さて，続いては，iモード対応HTMLの言語仕様を見ていきましょう。

先ほども簡単に触れましたが，この「iモード対応HTML」はHTML3.2，4.0を基に，携帯端末など用に策定された「CompactHTML」を参考にして作られたHTML言語仕様です。基本的に，この言語はコンパクトにしたHTMLそのものですから概念，文法もHTMLのそれがそのまま使われています。<HTML>～</HTML>で囲われた部分にヘッダ部分，エンテティボディ部分が入り

```
<HTML>
        <HEAD>
                ヘッダ部分
        </HEAD>
        <BODY>
                コンテント部分
        </BODY>
</HTML>
```

という形を取ります。

その中にたとえば，ハイパーリンクのA HREFタグであれば<A HREF ="bar.html">bar.htmlへのリンク</A>のように書かれるわけですね。では，通常のHTMLとの違いは……と探していくとこんなところがあります。

・絵文字の出力方法

iモードでは「絵文字」と呼ばれる，特殊なキャラクタを使用することができます。これにはたとえば，天気を表すときに使えそうな「晴れ」や，あるいは電話のボタンを表す［1］などがそうです。これらのキャラクタと文字コードの対応表はNTT ドコモのWebサイトに一覧があります。

http://www.nttdocomo.co.jp/mc-user/i/tag/emoji/index.html

これらの絵文字はシフトJISコードの，ほかの漢字の存在しないエリアに割り振られています。この絵文字をiモード用HTMLファイルで表示させ

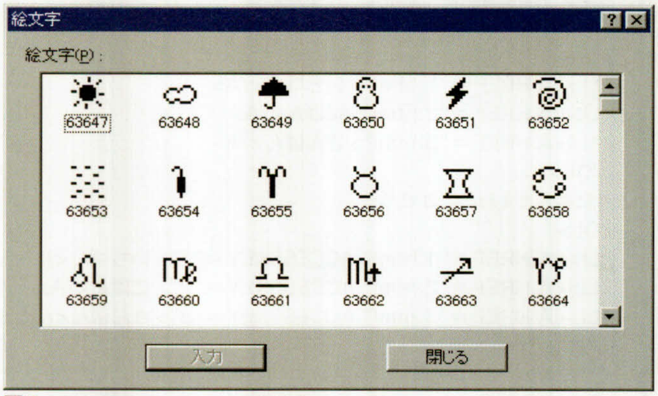


図3　こちらはiホタルパーソナルの絵文字挿入メニュー

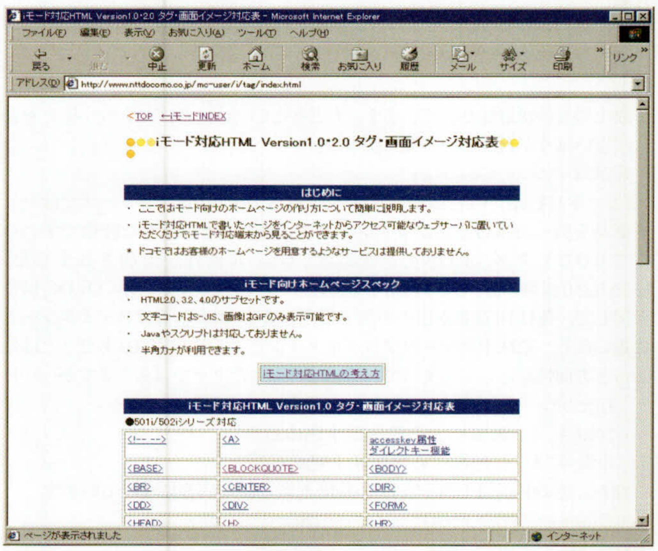
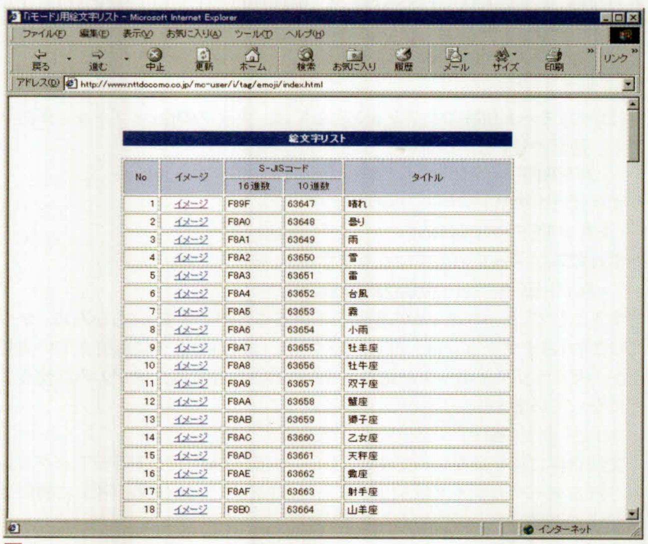


図2　NTTドコモのWebサイトにある「iモード対応HTML Version1.0/2.0 タグ・画面イメージ対応表」ページ



No	イメージ	S-JISコード 16進数	S-JISコード 10進数	タイトル
1	イメージ	F89F	63647	晴れ
2	イメージ	F8A0	63648	曇り
3	イメージ	F8A1	63649	雨
4	イメージ	F8A2	63650	雪
5	イメージ	F8A3	63651	雷
6	イメージ	F8A4	63652	台風
7	イメージ	F8A5	63653	霧
8	イメージ	F8A6	63654	小雨
9	イメージ	F8A7	63655	牡羊座
10	イメージ	F8A8	63656	牡牛座
11	イメージ	F8A9	63657	双子座
12	イメージ	F8AA	63658	蟹座
13	イメージ	F8AB	63659	獅子座
14	イメージ	F8AC	63660	乙女座
15	イメージ	F8AD	63661	天秤座
16	イメージ	F8AE	63662	蠍座
17	イメージ	F8AF	63663	射手座
18	イメージ	F8B0	63664	山羊座

図4　NTT DoCoMoのサイトにある絵文字一覧

るには，DoCoMoホームページの「絵文字リスト」に表示されているS-JISコードを直接HTML中に書き込みます。

例: 「晴れ」の絵文字を表示する

```
&#63467;
```

なお，「ｉモード対応HTML Version1.0・2.0 タグ・画面イメージ対応表」のAタグの解説では，

```
<A HREF＝"http://www.nttdocomo.co.jp/tokyo.htm"
ACCESSKEY＝"1">[1]東京</A><BR>
```

絵文字を直接HTML文書に記入するかのように書かれていますが，テキストエディタなどで記入するには，

```
<A HREF＝"http://www.nttdocomo.co.jp/tokyo.htm"
ACCESSKEY＝"1">&#63879東京</A><BR>
```

とすることで，ｉモード端末上には，

```
[1]東京
```

と表示されます。

・アクセスキー機能

ｉモード対応HTMLで特徴的なのは<A>タグとハイパーリンクやフォーム中のデータ記入・選択部分を占める<INPUT>タグに「accesskey =""」で表現する，アクセスキーオプションがあることです。たとえば画面中に，

```
1.いちばん
2.にばん
3.さんばん
```

と表示するとしましょう。アクセスキーを使うと，十字ボタンで選択肢を選んでから，(i)ボタンでそれを選択しなくても「1」や「2」のキーを押すことで，ワンタッチで選択ができるようになります。

通常のHTMLであれば，

```
<OL>
<LI><A HREF＝"1.html">いちばん</A>
<LI><A HREF＝"2.html">にばん</A>
<LI><A HREF＝"3.html">さんばん</A>
</OL>
```

のように書きますね。ここで，

```
<OL>
<LI><A HREF＝"1.html" ACCESSKEY＝"1">いちばん</A>
<LI><A HREF＝"2.html" ACCESSKEY＝"2">にばん</A>
<LI><A HREF＝"3.html" ACCESSKEY＝"3">さんばん</A>
</OL>
```

と，それぞれのハイパーリンクの<A>タグにACCESSKEYオプションで「0」～「9」あるいは「※」「#」のボタンを設定することで，電話機に表示したページの選択肢をワンタッチで選ぶことができるのです。

ちなみに，このACCESSKEY =""はInternetExplorerやNetscape Navigatorでは無視されるだけですので，通常のホームページを書くときに使用してもなんら差し障りありません。

・telオプション

ほかのｉモード独特のオプションとしては，Aタグのtelオプションがあります。通常のHTMLでは，

```
<A HREF＝"http:
```

とすればほかのWebページへのハイパージャンプ，

```
<A HREF＝"mailto:
```

とすれば送信メールの編集になりますが，ｉモードでは，

```
<A HREF＝"tel:090xxxxxxxx">
```

とすることでｉモード端末から電話をかけることができます（ちなみに，かつてはこのtelオプションは一部のブラウザではほかのタグにも使えていましたが，セキュリティホールが見つかったため，現在ではAタグのみに使うことになっています）。

・INPUTタグの制約

文法自体に違いはないのですが，ｉモード用HTMLではINPUTタグで入力されるデータのサイズは512バイト，SELECTでは4096バイトに制限されています。

・FROMのGETメソッド

ｉモードでは，ブックマークに入れられるURLの長さは1URLあたり100バイトです。ですので，URLエンコード後の長さは必ず100バイト以内でなければなりません（それ以前に，100文字ものURLを携帯電話で入力するのは苦行以外の何者でもないと思いますが…）。

Version2.0の仕様

ｉモード用HTMLには501iシリーズで使えるVersion1と502iシリーズで使えるVersion2があります。第1世代機であるF501で使用できるのがVersion1。その後，F502シリーズ用に拡張されたり，いくつか修正されたのがVersion2です。Version2はVersion1とは上位互換があります。つまり，ｉモード対応HTML Ver1.0のみで作成したページは501/502両方で同じように表示されます。ｉモード対応HTML Ver2.0で追加されたタグ，属性は501iシリーズで表示した場合は無視されます（るはずです。DoCoMoの資料では「基本的に」となっているので，もしかしたらこの原則から外れるものがあるのかもしれませんが，いまのところ確認できていません）。

Ver2.0拡張部分

ｉモード対応HTML Ver2で追加されたタグは以下のとおりです。

- MARQUEE 文字列の画面シフト表示機能
- BLINK 文字列の点滅
- SELECT～OPTION 選択肢折返機能
- META

METAタグで書くことができるのは，

```
<META http-equiv＝"Content-Type" content＝"text/html ;
charset＝SHIFT_JIS">
```

と固定されています。Refreshなどを使うことはできません。なお，ｉモードでは「ｉモード対応HTML Version1.0・2.0 タグ・画面イメージ対応表」ではシフトJISコードで書かれたHTMLファイルしか表示しないことになっていますが，実際にはｉモード端末はJISコード(ISO-2022-JP)で書かれたHTMLも正しく表示することができます。

```
<META charset＝SHIFT_JIS">
```

で文字コードにシフトJISコードが指定されていて，実際にはJISコードで書かれているページはというと，これも正しく表示できるようです（もしかして，METAタグの中身は無視しているだけかもしれません）。

- </P>
- </DT>
- </DD>
- </LI>

以前のバージョンの「ｉモード対応HTML Version1.0・2.0 タグ・画面イメージ対応表」にはなかったタグです。

以上のものが追加されています。もともとCompactHTMLの仕様を考えればないほうがおかしいのですが……。

・BODYタグのCOLOR属性

ｉモードHTML Version2対応の携帯電話つまりF502シリーズではカラー液晶を持った携帯電話がラインナップされたことが大きな特徴です。そこでBODYタグ，FONTタグにはCOLOR属性が追加されました。HTML2.0以降のBODYタグ，FONTタグのCOLORタグのCOLOR属性と同じで，色は16進数か色の名前で指定します（画面装飾はスタイルシートに追い出し，意味上のマークアップをメインとするHTML4.0あたりとはちょっと方向性が違ってしまっている感じの困ったタグではありますが……）。

```
bgcolor＝"16進数による色指定｜色の名前"
text＝"16進数による色指定｜色の名前"
link＝"16進数による色指定｜色の名前"
```

なお，色の名前はそれぞれ以下の16進数と同じ内容になっています。

```
Black＝"#000000"     (黒)
Gray＝"#808080"      (グレー)
Maroon＝"#800000"    (暗い赤)
```

表

属性値	モード1	モード2
1	全角かな	全角文字
2	半角カナ	半角文字
3	英字	半角文字(小文字推奨)
4	数字	半角文字(小文字推奨)

リスト1

```
<HTML>
<HEAD>
<TITLE></TITLE>
<LINK REL=stylesheet TYPE="text/css" HREF="pcbrowser.css">
<STYLE TYPE="text/css">
</STYLE>
</HEAD>
<BODY>
<H1>サンプルページ</H1>
<HR>
<H2>[このページは…]</H2>
<P>
このページは「iモードとPC用ページを同じCGIで表示する」ためのサンプルプログラム・
「イベントカレンダー」のインデックスページです(Oh!X 2000春号に掲載予定)。現在イベ
ントリストのテスト中ですので、よろしかったら、足跡代わりに適当になにかそれらしいイベ
ントを追加していってください。
<DIV ALIGN=CENTER><A HREF="eventlist.cgi">イベントリストを表示する
</A></DIV>
<DIV ALIGN=LEFT>なお、現在サンプルに書き込まれている<STRONG>データはすべて架
空のもので実際のイベントなどとは一切関係ございません</STRONG>のであらかじめご了承
ください。また、データを書き込まれるかたも、そのつもりで楽しい架空のデータを書き込ん
でください。(^^<BR>
また、書き込まれた<STRONG>サンプルデータは雑誌付録CD-ROMには収録されません
</STRONG>のであしからず。残念ですけどね。(^^;</DIV>
<HR>
このページ対するご意見・御感想などは<A HREF="mailto:de@sakuramail.com">筆者
</A>まで。<BR>
掲示板もこちらにあります<A HREF="http://web.pe.to/~deyamato/cgi-
bin/minibbs.cgi">(で)ページのminibbs</A>。さびれてますけど(すいません)。
<BR></P></BODY>
</HTML>
```

リスト2

```
BODY {
        /*ページ内の背景色*/
        BACKGROUND-COLOR: #fdfad2
}
H1 {
        BACKGROUND-COLOR: rgb(132,233,253);
        TEXT-ALIGN: center
}
H2 {
        BACKGROUND-COLOR: rgb(184,257,254);
}
H3 {
        BACKGROUND-COLOR: #eed
}
H4 {
  font-size : medium;
  background-color : #b8edfe;
  width : 400px;
}
HR {
  background-color : transparent;
  background-image : url("image2.gif");
  background-repeat : repeat-y;
  background-attachment : fixed;
  background-position : center center;
  border-style : none;
  width : 600px;
  height : 8px;
}
P {
        TEXT-INDENT: 5em
}
P.noindent {
        TEXT-INDENT: 0em
}
BLOCKQUOTE {
        COLOR: #030; FONT-SIZE: small
}
A.normref {
        COLOR: red
}
A.informref {
        COLOR: green
}
STRONG {
        FONT-WEIGHT: bold
}
```

Purple ="#800080"　(暗い紫)
Green ="#008000"　(くらい緑)
Olive ="#808000"　(暗い黄色)
Navy ="#000080"　(ネイビーブルー)
Teal ="#008080"　(タール色)
Silver ="#C0C0C0"　(少し明るいグレー)
White ="#FFFFFF"　(白)
Red ="#FF0000"　(赤)
Fuchsia ="#FF00FF"　(紫)
Lime ="#00FF00"　(明るい緑)
Yellow ="#FFFF00"　(黄色)
Blue ="#0000FF"　(青)
Aqua ="#00FFFF"　(水色)

・FONTタグのCOLOR属性

BODYタグと同じくFONTタグにもCOLOR属性が追加されています。BODYタグと同じように16進数，または名前での色指定です。

color ="16進数による色指定 | 色の名前"

・LIタグのtype属性

・OLタグのtype属性

番号つきリストのラベル部分を算用数字(デフォルト)，小文字アルファベット，大文字アルファベットのうちから選びます。HTML 2.0のOLタグのtype属性とまったく同じです。

type ="1 | a | A"
start ="開始番号"
　1:算用数字にします
　a:小文字アルファベットにします
　A:大文字アルファベットにします

また，start属性は数値で指定し，ラベル形式にあった文字が表示されます。

・INPUT type="text"とTEXTAREAタグのistyle属性

TEXTAREAはブラウザを見たユーザーが文字を記入するわけですが，このistyle属性は文字を記入する場合の電話のキーの初期状態(つまり，かな文字入力であるのか，あるいは数字であるのかなど)を決めます。

このistyle属性は端末(電話)の設定が「かな入力」になっていたか，ポケベル入力かで微妙に設定が異なります。NTTドコモのページ中の表示に「モード1」と書かれているのが「かな入力」，「モード2」と書かれているのが「ポケベル入力」を意味しています。

つまり，もし，電話側がかな入力モードで下のようにistyle属性に4が指定されていたら，電話上で押されたボタンの数字がそのまま入力欄に入ります。

<INPUT type ="text" name ="phone_number" istyle ="4"
size = 13 maxlength = 13>

iモードとパソコンの両方で綺麗に見えるページ

さて，ここまで解説してきたことで，すでにHTMLの知識をお持ちの方なら，簡単にiモード用のWebページを作ることができることでしょう。ところで，上の要領でiモード用のホームページデータを作ってみると，非常にそっけないものになってしまいますね。CGIでiモード，その他と出力を分けてしまう(つまり，それぞれ用にHTMLコンテンツを出力してしまう)というのもそれはそれで有効なのですが，もし，同じに作っておいて，どちらでもそれなりに綺麗に見える，使い回しの利くコンテンツを作ることができれば，CGIで書くコードも少なくて済みますので，同じコンテンツで，それなりにどちらでも綺麗に見えるコンテンツの作り方も少しだけ考えておきましょう。

綺麗にするといえば，いま，よく使われるHTML3.2レベルのコンテンツでは，<TABLE>タグを使った表組みレイアウト，あるいは，セルに背景色をつけたり，<FONT COLOR =やSIZE =でフォントの色やサイズをつけたりといったあたりが多く使われていますね。が，この辺をiモードでそ

のまま使おうとすると，いくつか問題が出てきます。

まず，HTMLコンテンツのサイズが膨らんでしまうこと。iモードで表示できる1ページのサイズはわずか5KBですから，いらないタグでファイルのサイズが大きくなるのは問題です。それから，iモード対応HTMLではこれらの字飾りの対応はちょっとハンパで，たとえば，表のセルの背景の色は変えられませんから，セルと文字の色を変えようとすると文字の色だけ変わってしまった（背景に字の色が埋まってしまって見えない）なんていうことも起こりえます。ということで，ページを修飾する部分だけを別にファイルにして追い出してみました。つまりCSS（カスケーディングスタイルシート）を使ってみます。

CSSに関してはもうご存じの方も多いでしょうね。CSSはHTMLやXMLで書かれた文書のデザイン面を定義するためのもので，HTMLのタグや要素のそれぞれの見かけをここで設定すると，スタイルシートに対応したブラウザではその部分が飾られて見える，というものです。たとえば，

<H1>タグはセンタリングして，文字背景を水色に，文字のサイズは特大に

というふうに定義すれば，HTML中に，

<H1>iモード用CGI</H1>

と書かれていたら，

iモード用CGI

というふうに表示されます。

このCSSはHTMLコンテンツ中で定義することもできますが，外部スタイルファイルとしてスタイルシート部分だけ独立したファイルとして追い出すこともできます。この場合，HTMLコンテンツ自体は<HEAD>～</HEAD>中で，

<LINK REL＝stylesheet TYPE＝"text/css" HREF＝"スタイルシートファイル名">

と書くだけです。iモード用HTMLには<LINK>タグは存在しませんから，

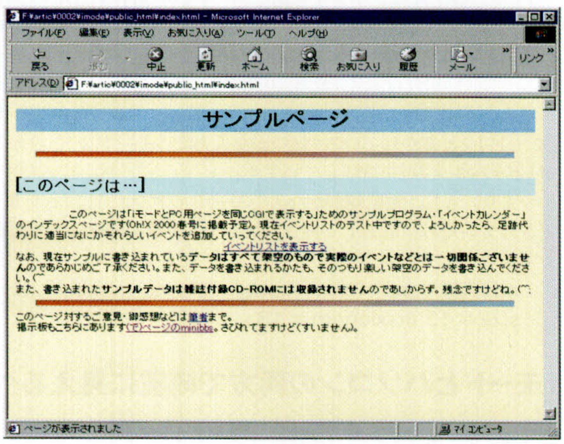


図5　iモードでもそれなりに見られるようにスタイルシートを使って作ったページ。PCで見ると，ちょっとそっけないだけのWebページだけど

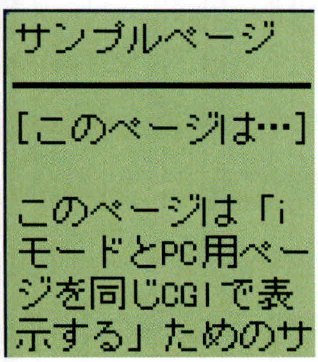


図6　iモードで見てもちゃんとこのとおり

ここは無視されるだけです。

たとえば，そこで作ってみたのが，

sample1.html

と，

pcbrowser.css

です。

sample1.htmlはごく普通のiモードHTMLに対応したページデータです。ただし，4行目で，

<LINK REL＝stylesheet TYPE＝"text/css" HREF＝"pcbrowser.css">

と外部スタイルシートを呼び出しています。

外部スタイルシートpcbrowser.cssでは，

BODYタグ：背景色 ＝ 肌色
H1タグ：背景色 ＝ 水色
　テキスト配置 ＝ 中央寄せ
　　：
　　：
HRタグ：背景：透過色
　背景イメージ：iamge2.gif
　　：
　　：

と各タグに大きさ，色，使用する画像，といったものを定義しています。

iモード携帯端末では，電話機のブラウザはスタイルシートを解釈しないでテキストだけを表示します。PCのブラウザは，スタイルシートを解釈できるものであれば，色や絵をつけて表示します。

つまり，このようにしておけば，PC用のブラウザでは飾られて，iモードでは文字だけが表示されて，便利だね，というわけなのです。もし，仮にPC用のブラウザでもCSSに対応してなければ，綺麗ではないですが，少なくともiモードと同じ程度には見ることができるはずです。

なお，当たり前の話ですが，このようにPC，iモード両用のコンテンツを作るのであれば，見栄えやイメージ（画像）に頼るようなデザインは避けたほうがいいでしょう。たとえば，表組みでページ内のメニューが作られていたり，タイトルが画像で作られていたりということですね。テーブルをレイアウトやメニューに使ってもiモードのブラウザは横スクロールができませんから表示できない，メニューの場合はそのページに飛ぶことができない，という自体に陥ってしまいます。また，イメージの場合，iモードで表示できるイメージは94×72ドットが最大の大きさですから，それより大きくなりがちなPC用のブラウザでは小さすぎますし，無理に拡大表示しようとすると汚くなってしまい，そもそもイメージを使った意味がなくなってしまいますし，イメージの多用は回線速度の遅いiモードでは表示に時間がかかり，いらいらさせられてしまうでしょう。

また，同じiモード用HTMLにはHTML3.2や4.01と共通のタグも多く存在しますが，同じタグをiモードとPC用ブラウザに表示させたとき，見栄えが同じになるわけでもないことにも気をつけてください。たとえば，先ほどの見出しの<H1>や<H2>タグですが，iモードでは文字の大きさは通常の文字と同じサイズ，見かけです。大きくなりませんし，太くもなりません。iモード，PC両用のコンテンツを作るのであれば，必要な情報はなるべく文字で伝えること，スタイルシートもHTMLでの文字の飾りもあくまで飾りであって本質的なものではない，と割り切るべきでしょう。

当然ですが，あくまでスタイルシートでつけることができる飾りは「飾り」であって，それ自体が見る人に必須の情報であってもいけません。だって，それではiモードでは見えませんからね。

iモード用CGIを作る

さて，スタイルシートを使うとHTMLコンテンツ自体はiモードでもPCでもそれなりに見られるものを作ることができますが，動的なコンテンツを作る場合にはそれだけではなく，ほかの処置も必要になります。なぜなら，スタイルシートはHTMLの見かけの問題は解決してくれますが，ファイルサイズや文字コードの問題は解決してくれないからです。

ここでは，iモードで使えるCGIを作る際に必要なことを説明しておきましょう。

・Content-Length：が必須

NTT ドコモのホームページの説明にあるように，iモード対応のCGIではHTTPヘッダに，コンテンツのサイズを意味するContent-Length:が必須となっています。CGIでは，HTMLコンテンツを標準出力に出力することで，ブラウザにHTMLコンテンツを表示させることができますが，その前処置としてHTTPヘッダをいくつか出力します。そのときに，

```
Content-Type: text/html
Content-Length: $strlen
```

の2行が最低限必要，ということになるわけです。つまり，「ページを出力するときに，出力するコンテンツは確定していて，そのサイズがわかっていないといけない」ということですね。

なお，普通にHTMLコンテンツを出力するとき以外にも，この制約はつきまといます。たとえば，Location:を使ってとかのページ，たとえばwww.foo.barに飛ばそうとしてもHTTPヘッダは，

```
Location: http://www.foo.bar¥n¥n
```

とするだけではその場所に飛ばすことはできません。Locationでほかに飛ばすには，

```
Content-Length: 0
Location: http://www.foo.bar¥n¥n
```

としてください。

・環境変数HTTP_USER_AGENTは"DoCoMo"で始まる。

CGIプログラムからiモード携帯端末を判断するにはUser-Agent変数を参照するといいでしょう。iモードではUser-Agentにはこのような文字列が入ります。

```
UserAgent: DoCoMo/1.0/F501i
UserAgent: DoCoMo/1.0/F502i/c10
```

左から順に「DoCoMo」の文字列，「HTTPバージョン」，「機種名」。F502シリーズの場合はそのあと「cXX」と読み込み可能な画面のキャッシュ最大値が入ります。とりあえず，

「^DoCoMo」

でパターンマッチングすればいいでしょう。少なくとも，現行のiモード端末はすべてこれにマッチします。ほかの関係ないUserAgentがヒットしないか，ということですが，同じDoCoMo製品でも，たとえば，ブラウザボードなどは，

```
sharp pda browser/4.5 [ja] (BrowserBoard/1.0)
```

という文字列が表示されるようなので，おそらく，ほかのDoCoMo製品もiモード以外でこれにマッチするものはない……ということにしておきましょう。

・iモードには半角カナがある

忘れがちなのがこれです。iモードでは半角カナ文字を使いますが，通常，PCのブラウザなどで見ている場合にはこれは避けるべきでしょう。jcode.plには，

```
&jcode'h2z_xxx (*line) ;
&jcode'z2h_xxx (*line) ;
```

といった半角←→全角のコンバージョン関数があるので，これを使ってそれぞれ適切な文字コードで出力してください。

共通CGIの骨

さて，ということで作ってみたのがリスト3。iモードとそのほかのブラウザどちらにもそれなりに見える表示をするためのCGIのスケルトンです。このプログラムでは（私のプログラムには珍しく）Perlモジュールではなく，cgi-lib.pl，jcode.plとCGIでよく使われるパッケージを使用してみました（普通にインターネットサービスプロバイダのホームページ領域を使ってCGIを使っている場合は，このほうが通用しやすいだろう，というのがその理由です。本当はCGI.pmなりが使えればServer Errorのメッセージ表示などもできてデバッグも楽なんですが……）。

・1～17行あたり　下準備

・18～30行あたり　各種変数設定

まずは，cgi-lib.plのReadParse()を呼び出して，QueryStringsを分解して，変数などをハッシュに登録します。この辺は通常CGIを作るときと同じ手順でしょう。18～29行でQueryStringsを分解しています。FORMを使っていて，GETメソッドで呼び出したときも，ここで変数の内容が設定されます。

・31～41行あたり　POSTデータのデコード

31～41行では，POSTメソッドで呼び出されたとき。つまり，このスケルトンはGET，POSTどちらのメソッドで呼び出されたときでもINPUTタグで設定された変数を読み出すことができるように，両方のデコード部を持っているのです。実際にCGIを，これを使って書くときにはGET，PUTどちらかいらないほうは削ってしまってかまいません。GETの際には変数は%qsに入っています。POSTの場合は%postvalに収められています。

・43～48行　HTMLヘッダの作成

さて，43行～48行目では，HTMLコンテンツを一時的に収めておく変数$strbufにHTMLコンテンツのヘッダ部分として，

```
<HTML>
<HEAD><TITLE>$title</TITLE>";
<META HTTP-EQUIV="Content-Type" CONTENT=
"text/html; charset=sjis">
</HEAD>
```

といった文字列を変数に入れています。要するにホームページのタイトルと，このページがシフトJISコードで書かれているよ，ということをいっています。

・50～54行　ユーザーエージェントの判定

51行目では，このCGIをアクセスしたユーザーエージェントがiモードであるかどうかを判断し，そうである場合には$uaという変数に"i"という文字列を代入しています。iモードであるかどうかの判断には，

```
^Docomo
```

でパターンマッチングを行っています。

・56～67行　背景，スタイルシートの設定

HTMLコンテンツのボディ部分の始まりである<BODY>を設定していますが，もし，PC用のページ出力で背景画像を設定したい場合はここで設定する必要があります。コメント化されている部分がそれです。ユーザーエージェントを判定して，iモードでない場合には外部スタイルシートと，背景画像を設定しています。

・69～73行 ToDo

ここで実際に表示するHTMLコンテンツを作るためのコードを書きます。iモードとそれ以外と基本的に同じHTMLでいい場合（先に書いたように，同じHTMLでもスタイルシートを活用してそれなりに見せる方法もありますし）はひとつのコードで済みますが，違う機能を作る場合には50～54行で確認したユーザーエージェントによって場合分けして複数のコードを書く必要があります。

・75～78行　作成したコンテントの出力

ここまで作成してきたHTMLボディとHTMLコンテントそのものを，

```
</BODY>
</HTML>
```

を締めます。サブルーチン「print_content」で表示がされて，CGIプログラムは終了，ということになります。

・81行～　実際の出力ルーチン

ある変数（$strbufとしましょう。文字列のバッファというか緩衝，という意味でつけてみました）に入られた文字列が，最後にこの文字列の長さContent-Length;と，本文$strbufが標準出力に出力されます。

```
#コンテンツの表示
    my ($strbuf, $ua) = @_;
    if ($ua eq "i") {
            $strbuf = jcode'z2h_sjis ($strbuf)
    }else{
            $strsuf = &jcode'sjis ($strbuf) ;
    }
    my $strlen = length ($strbuf) ;
```

```
    print "Content-Type: text/html¥n";
    print "Content-Length: $strlen¥n";
    print "¥n";
    print "$strbuf¥n";
}
```

jcode'sjisは与えられた文字列をシフトJISに変換して返します。つまり，与えられたHTMLコンテンツをシフトJISに変換し，HTTPヘッダとして，

```
Content-Type: text/html
Content-Length: HTMLコンテンツの長さ
```

を出力してから，先ほど，シフトJISに変換したコンテンツを送信しているわけですね。

なお，jcode.plの性質としてjcode'sjisやjcode'eucで変換したコンテンツは，文字列の途中などで変換に失敗すると，そのままなにも変換作業などをしないで元の文字列を返してしまいますので，注意が必要です。理想的には，コンテンツが正しくシフトJISに変換されているかどうかは確認するようにプログラムを追加しておいたほうがいいでしょう。ちなみに，jcode.plでなく，Jcode.pmを使った場合はそのような現象は起きないようです。

実際に使ってみる

さてさて，で，このスケルトンを使ったCGIでスタイルシートを使って，どちらでも使えるようなページを出力する，という合わせ技で作ったのが，今回のサンプルプログラム「eventlist.cgi」です。

このプログラムはiモード，PC用コンテンツをひとつのCGIで出力することができます。表示されるのは，簡単にいうと「予定表」のようなものです。たとえば，

```
2/29 Oh!X 2000春号発売
4/30 夏号マスターアップ
6/20 夏号発売
```

というように予定の一覧表を表示するCGIプログラムです。実際には，予定というよりは，たとえば，なにかイベントの一覧表とかそんなものを想定して，入力できる項目は少し数が多くなっていますが。イベントの名前，日時，場所，イベントの説明，詳細/連絡先を入力して，

```
-----------------
イベントの名前: Oh!X発売日
日時：2000/02/28
場所: 全国の書店にて
イベントの説明: このサンプルプログラムが掲載される，Oh!X
2000春号が発売されます。
詳細/連絡先 ソフトバンクパブリッシング
-----------------
```

ということが表示できます。また，イベントの入力時に入力したパスワードを入力することで一度入力したイベントを削除できるようにもしています。ま，「掲示板」プログラムのちょっと変わったやつ，とでも思えば間違いないでしょう。

さて，iモードの場合，通常のホームページを見ると，「容量が大きすぎると読み込みができない」「画面が狭いため，表示できる程度でも文字数が多

リスト3

```
#!/usr/local/bin/perl
#
# iskel -  i-mode用CGIのスケルトン
#
#     by de.yamato 1999,2000
#      e-mail de@sakuramail.com
#必要なモジュールの読み込み宣言
require "cgi-lib.pl";
require "jcode.pl";
# cgi-libを読み出して、オプションなどの設定
&ReadParse(*input);
#このページのタイトル
my $title = "iモードスケルトンCGIページ";
# $strbuf(ページに表示する内容が入る変数)を定義
# HTMLコンテンツのヘッダ部分を定義
my $strbuf = "<HTML>\n<HEAD><TITLE>$title</TITLE>";
$strbuf .= "<META HTTP-EQUIV=\"Content-Type\" ";
$strbuf .= "CONTENT=\"text/html; charset=sjis\">";
$strbuf.="</HEAD>\n";
$strbuf.="<BODY>\n";
# -- テスト用  全ての環境変数表示
$strbuf .= "<h2>環境変数</h2>\n";
$strbuf .= "<dl compact>";
foreach $key (sort keys %ENV) {
     $strbuf .= "<dt><b>$key</b></dt><dd>$ENV{$key}</dd>\n";
}
$strbuf .= "</dl>";
# ユーザーエージェント変数「$ua」 iモードなら"i"
my $ua;
if($ENV{"HTTP_USER_AGENT"} =~ /^DoCoMo/){
     $ua=i;
}
# QUERY_STRINGのデコード (%qsに内容が入る)
$qs = $ENV{"QUERY_STRING"};
@qs = split(/&/,$qs);
foreach $cnt (0 .. $#qs){
        $qs[$cnt] =~ s/\+/ /g;
        $qs[$cnt] =~ s/%(..)/pack("c",hex($1))/ge;
        ($param,$val)= split(/=/,$qs[$cnt],2);
        $qs{$param} = &jcode'h2z_euc($val);
}
#POSTの場合のデータのデコード
@val = split(/&/,$input);
foreach $i(0 .. $#val){
        $val[$i] =~ s/%(..)/pack("c",hex($1))/ge;
        ($name,$value) = split(/=/,$val[$i],2);
        $value =~ s/\+/ /g;
        $postval{$name} = &jcode'h2z_euc($value);
}
# (例えば http://~/imode_skel.cgi? page=20 だったら$pageに20が入
るよ
#うに)
# $page = $qs{"page"};
## *** To Do *** -ここからプログラム本体を作る
## (print の代わりに $strbuf .= ""すること)
## *** ENDOF To Do ***
#コンテンツ内容の表示
$strbuf .= "</body></html>\n";
&print_content($strbuf,$ua);
exit(0);
sub print_content {
#コンテントの表示
     my ($strbuf,$ua) = @_;
     if($ua eq "i"){
             $strbuf = &jcode'z2h_sjis($strbuf);
     }else{
             $strbuf = &jcode'sjis($strbuf);
     }
     my $strlen = length($strbuf);
     print "Content-Type: text/html\n";
     print "Content-Length: $strlen\n";
     print "\n";
     print "$strbuf\n";
exit(0)
}
```

いとスクロールが長くなって不便」という問題が出てきます。そこで，掲示板や今回のようなプログラムの場合，表示方法になんらかの工夫をしなくてはならないでしょう。そこで，プログラムを作る前に，まず，どのように画面を表示するのか簡単に考えておいたほうがいいでしょう。たとえば，今回のイベントカレンダーの場合，ふつうにPCに表示させるのであれば，

```
-----------------
イベントの名前: Oh!X 発売日
日時：2000/02/28
場所: 全国の書店にて
イベントの説明: このサンプルプログラムが掲載される，Oh!X
2000 春号が発売されます。
詳細/連絡先 ソフトバンクパブリッシング
-----------------
-----------------
イベントの名前: テストイベント2
日時：2000/05/05
場所: こどもの日イベント
イベントの説明:
詳細/連絡先
-----------------
      ：
      ：
```

というふうにイベントの詳細な紹介を10か20続けて表示してもいいでしょう。

ですが，iモードの場合， 表示できる領域はたとえば，私の持っているF502iの場合で全角で8×7文字分程度です。画面は上下にスクロールできるにせよ（iモードのブラウザは左右にはスクロールできません。これはiモードのWebブラウザの仕様です），あまり分量のあるコンテンツには向かないでしょう。今回であればイベントひとつ分の情報がちょうどいいくらいなのではないかと個人的には思います。

そこで，今回は，iモードでアクセスした場合には，他のブラウザでアクセスした場合と挙動を変えて，イベントリストのダイジェストだけ表示するようにしましょう。イベントリストの項目の中で文字数が多くなりそうなのは「イベントの説明」の部分でしょうから，ここの部分を省いて，

```
(1) Oh!X 発売日
日時：2000/02/28
場所：全国の書店にて
(2) テストイベント2
日時：2000/05/05
場所：
```

とだけ表示するようにします。そして，アクセスキーの(1)や(2)が押されると，イベントの紹介の全文が表示されるようにしてみましょう。図で書くと図nのようになりますね。

ということで，このCGIの表示では，

- iモードのときの1ページ詳細表示
- iモードのときのリストダイジェスト表示
- その他のブラウザのための表示

の3パターンの表示方法を用意すればいいことになりますね。

さて，先ほど作ったスケルトンから，このプログラムを作るために，変更を要するのは，<BODY>タグを作成する部分でiモード以外とiモードを分けるためにコメント化されていた部分を生かす以外では基本的に69行からの「To Do」部分です。今回の3パターンの表示方法もここに埋め込みます。そして，ユーザーエージェントがこのCGIにアクセスしたときの状況でもって，それぞれどのルーチンが動くかを場合分けすることにします。

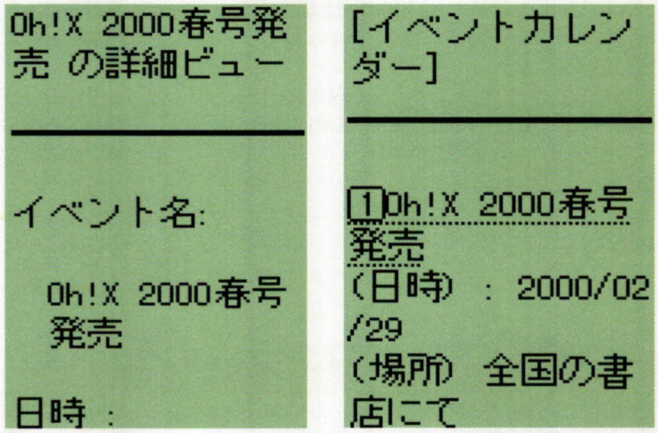


図8　iモードではリストの表示はダイジェスト。ボタンを押すと，その項目の詳細が表示される

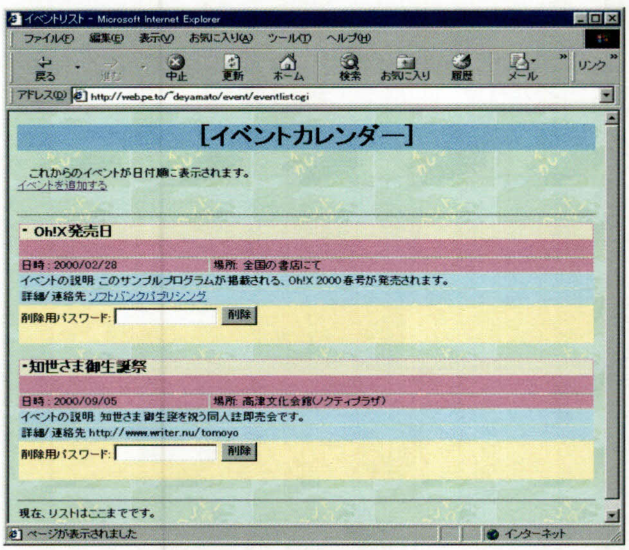


図7　Internet Explorer 5.0を使用した場合のイベントリストCGIの表示。タイトルなどはスタイルシートによる設定で色などの飾りがつけられ，リストはCGIによって詳細な情報が初めから表示される。発売日は気にしないこと

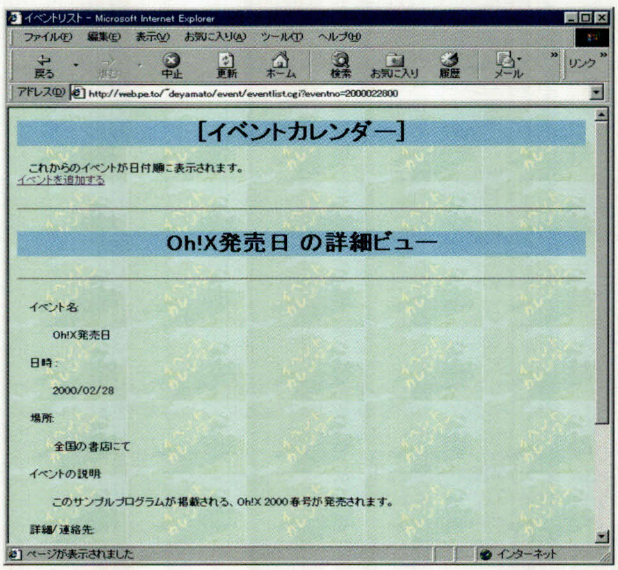


図9　PCで見てみたイベント詳細表示。スタイルシートは生きているので，背景や見出しの飾りも表示されている

まず，iモード用の1イベントの詳細表示ですが，ひとつのイベントを選んで表示するにはイベントの番号が必要ですから，リスト表示→1イベント表示する際に，なんらかの変数を渡すはずです。

とりあえず，このプログラムでは，

http://~/eventlist.cgi&eventno＝xxxx

と「eventno」というオプションを設定することで1イベント詳細表示をする，ということに決めました。残りはiモードのリスト表示とそのほかのリスト表示ですね。これは$uaが"i"であるかどうかでiモードかどうかがわかる，ということでした。ということで，最終的に，

```
if ($eventno ne "") {
  イベントの詳細表示;
}else{
    if ($ua eq "i") {
        iモード用リスト表示;
    }else{
    通常法リスト表示
    }
}
```

とすることにしてみました。

で，できたのがeventlist.cgiです。PCのブラウザで見ると表示される画面はPCだと初めから詳細の一覧になっているのが写真からもわかりますね。iモードでは，これがリスト画面では名称，日時，場所の項目だけが表示され，名称の番号のボタンを押すと，そのイベントの詳細な情報が見られるようになっています。

ちなみに，HTMLコンテンツの<HEAD>～</HEAD>部分の出力はどのページの出力でも同じなので，URLにeventno＝……と変数さえ設定してやれば，iモードでしか使わないページ，1イベントずつの詳細情報もPCで見ることができますので，興味のある方は試してみてください。たとえば，2000年3月21日の最初のイベントであれば，yyyymmddの00番目のイベント，ということで，

2000032100

というイベント番号が割り振られます。もし，このCGIがhttp://www.foo.com/eventlist.cgiなのだったら，

http://www.foo.com/eventlist.cgi?eventno＝2000032100

というURLを指定してやると，iモードでしか表示しない，1イベント＝1ページのモードでページが出力されるはずです。

さて，以上でiモード，PC両方でそれなりに見られるCGIが完成しました。いかがでしたでしょうか？　実際に使用するには，たとえば，スケルトン部分だけパッケージ化していろいろ使い回すなどされるといいかもしれません。

今回作成したサンプルプログラムは，筆者のホームページ中に，しばらく置いておきますので，もし，実際のCGIの動きを試してみたい方は，アクセスしてみてください。

http://web.pe.to/~deyamato/event/

(有効期間はOh!X春号が発売されてから1カ月程度とさせていただきます)。

参考文献

iモード対応 HTML Version1.0 2.0タグ・画面イメージ対応表
http://www.nttdocomo.co.jp/i/tag/

Another HTML-lintホームページ:
http://www.tecjapan.co.jp/htmllint/

OpenSpace本部 iMode HTMLタグ一覧（古籏一浩氏のwebサイト）
http://www.shiojiri.ne.jp/˜openspc/iHTML/index.html

iモードキャプチャ画面はdB-SOFT社のi.ホタルパーソナルを使いプレビューモード画面をキャプチャしたものです。

COLUMN サンプルプログラムの使い方

このサンプルプログラムはサンプルプログラム「eventlist.cgi」，iモードCGI用スケルトン「imode_skel.cgi」ほかをはじめとしたファイルがアーカイブされ，「eventlist.tgz」という名前で収録されています。これを解凍すると以下のようなファイルが作られます。

imode_skel.cgi …iモードCGI用スケルトン
eventlist.cgi … イベントリストCGIサンプルindex.html … このCGIの簡単な説明
pcbrowser.css … PCブラウザ用スタイルシート
eventbg.gif … PCブラウザ表示用の背景
viewdbm.pl … dbmファイルの内容を見るためのプログラム
private/addevent.cgi …イベント追加実行CGI
private/events.dbm … イベントデータファイル
private/newform.cgi … イベントデータ入力用フォーム出力CGI
private/inittable.pl … イベントデータ初期化プログラム

これらのファイルをそのまま使う場合は以下の要領で使用してください。

・telnetなどでtarが使用できるUNIX系サーバが使用できる場合

tarでアーカイブした時点でファイル属性などは適当に設定されていますので，そのまま解凍してください。

```
tar zxvf eventlist.tgz
```

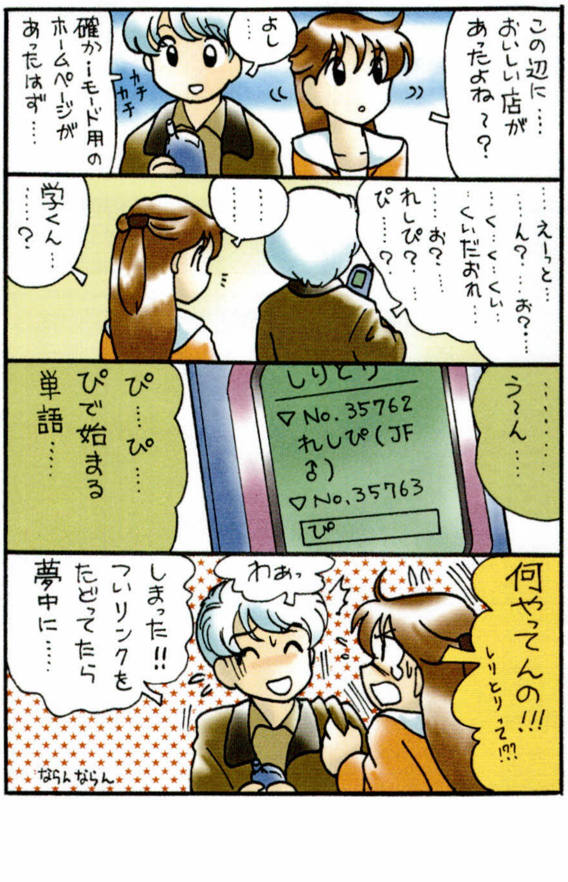
でeventというディレクトリが作られ，そのなかにプログラムファイルなどが展開されます。

・telnetなどは使用できない場合

telnetなどが使用できない場合は，いったん端末上でeventlist.tgzファイルを解凍することになります。winzipなど，tgz形式ファイルを解凍できるソフトウェアでファイルを解凍しておいてください。

できあがったファイルをftpクライアントソフトなどで転送したら，ファイルのファイル属性を変更します。ftpクライアントソフトなどで以下のファイルをそれぞれ指定された属性に変更してください。

eventlist.cgi … 755
private/addevent.cgi … 755
private/events.dbm … 666
private/newform.cgi … 755
private/inittable.pl … 755



Direct3D Retained Mode講座
六角大王ファイル→Xファイルコンバータを作る

菊地 功　Kikuchi Isawo

Direct3Dの高機能な部分を扱うのがRetained Modeの役目だ。しかし，3Dゲームを作ろうという場合には，そもそもの3Dオブジェクトを作成するという部分が難関として立ちはだかっている。既存のツールを使いつつ手間なくデータを作成する方法を考えよう。

Direct3D Retained Mode(RM)の入門も一段落して，今回は元々ここでゲームをやるつもりだったのだが，そっちは特集に回してしまったので(原稿はこれから書くんだが)，こっちはちょっと切り口を変えて軽めにいってみよう。ゲームを作るときに厄介なのはなにか。ゲームデザインとかコーディングとかいろいろあるが，力技だけではなかなかうまくいかないのがキャラクターデザインではなかろうか。特に3Dの場合は，方眼紙にスケッチして，その頂点を手で入力なんてシャレにならない。まあ，たいていの場合はなんらかのモデラを使うことになるだろう。3D CGソフトについてるやつとか。ただし，現状では，Xファイルを直接吐けるものはそれほど多くはない。そういうものは，コンバータなどを利用することになる。で，私はというと，

trueSpace3
myShade
DoGA L-1
六角大王

などが利用できる状態にある。trueSpaceはそれなりに大枚をはたいて買った割には，そのユーザーインタフェイスに馴染めず，お蔵入り状態であるので却下。myShadeはそこそこ使えるようになったつもりではあるので保留。DoGA L-1はそもそもゲームオブジェクトを作るにはポリゴン数が多くなりすぎる。六角大王は，実は今回までろくに使ったことがなかったのだが，単純な左右対称物ならば，かなりサクサク作れる。ゲームオブジェクト作りにはもっとも適しているようだ。

で，とりあえずは六角。当然のことのようにXファイルは吐けない(市販版の六角大王Super2はひょっとしたら吐けるのかも)ので，コンバータのお世話になる。六角のROKファイルからXファイルにコンバートするものは見つからなかったが，LightWave3DのLWOを経由すればオンラインソフトでまかなえる。が……ちょっといやーんなことに，きっちり両面を吐いてくれる。これはコンバータに限らず，直接Xファイルを吐くモデラでもいえることなのだが，両面吐くのは珍しいとしても，表裏を考慮してくれないものが多い。

そもそもモデラが想定しているレンダラは，ポリゴンに表も裏もない。両面描く。それが当たり前だからだ。それに対し，Direct3Dのようなリアルタイム系はレンダリング速度を優先するため，片面(Direct3Dの場合は頂点が時計回りになる方向)しかレンダリングしない。だもんで，多くの場合はポリゴンの表と裏が入り乱れたばらばらのオブジェクトが生成される。これはtrueSpaceのほうとそのコンバータを使用しても同じことだ。だからといって，両面吐かれてしまっても，ポリゴン数を減らして軽くしたいゲームオブジェクトにとっては迷惑なことだ。うむぅ……。

そのまま変換してみる

いずれにしても，ROKからXへコンバートするのに，いちいちLWOを経由するのは面倒なので，直接変換するコンバータを作ってみる。コンバータを自分で作ってしまえば，表裏問題は最悪でも手動で指定させるということも可能だからだ。Xファイル側のフォーマットは特に知る必要はない。IDirect3DRMMeshBuilder3::Save()メソッドを使えば，一発でファイルに落ちる。となると，問題はROKフォーマットと，それからMeshBuilderオブジェクトを作る部分である。

ROKファイルは，テキストエディタなどで開いてみるとわかるが，中身はテキストである。が，改行は0x0d 0x0aではなく，0x0dのみであるので注意が必要だ。要はバイナリモードでファイルを開き，改行は自分で判別しなければならないということになる。ま，たいした問題ではないが。で，フォーマットである。最初の1行は"ROKU4"であり，ファイルの識別子となっている。その後，軸や視点などの座標が書かれているようではあるが，必要なのは，"POINT"と"LINE0"，"FACE0"の文字列で識別される領域である。あと，フェースに色を設定するのであれば，"PALC0"の部分もだ。"POINT"領域には，頂点の情報が以下の順で頂点数分格納されている。

頂点番号(1から)
グループ0
表示状態
予約
対応する対称頂点番号
頂点座標X
頂点座標Y
頂点座標Z

適当に自分で解析した範囲であるので，間違いもあるかもしれないが，ここで必要なのは頂点座標のみである。同様に"LINE0"には稜線情報が，

頂点番号1
頂点番号2
グループ
表示状態

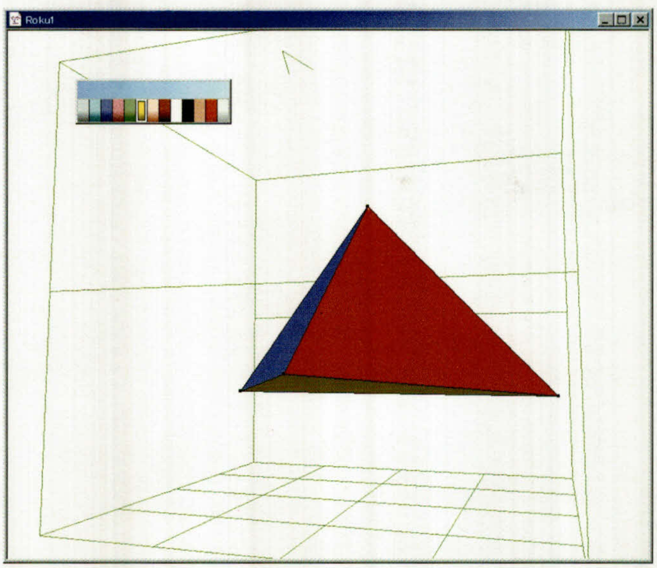


図1　六角大王でのモデリング

の順で格納されている。頂点番号は1からのインデックスで記述されているので，読み込んだときに1を引いておいたほうが混乱しないだろう。"FACE0"はフェース情報で，

稜線数
パレット番号
面番号
稜線番号１
稜線番号２
稜線番号３
稜線番号４(あれば)

となっている。ROKでは，フェースを頂点ではなく，稜線で記述されているのがミソだ。

六角大王では面は三角ないしは四角なので，稜線数（＝頂点数）は3または4になる。また，ここでも稜線番号は1からのインデックスとなっている。が，パレット番号はそのあとに出てくるパレット情報の0からのインデックス値である。で，そのパレット"PALC0"はというと，先頭にパレット数と不明な値があり，次いで，

明るいところの赤要素
明るいところの緑要素
明るいところの青要素
暗いところの赤要素
暗いところの緑要素
暗いところの青要素
予約

である。

明るいところと暗いところとは，六角大王の表示で使われるシェーディングの色範囲であり，色そのものとしては平均した中間色としていいだろう。図1のような三角錐を記述したROKファイルの生データをリスト1に表記しておくので，対応づけて見てもらいたい。

さて，これをメッシュに変換するのだが，MeshBuilderではフェースごとに色を個別に設定することができない。そのため，IDirect3DRMMeshを使い，色ごとにグループ(ROKのグループとは別)分けしてメッシュを作る。IDirect3DRMMeshBuilder3には新規IDirect3DRMMeshオブジェクトを作るためのメソッドCreateMesh()が用意されており，そちらでメッシュを作成したらIDirect3DRMMeshBuilder3::AddMesh()するという形になる。

```
IDirect3DRMMeshBuilder3 * builder;
lpD3DRM->CreateMeshBuilder( &builder );
IDirect3DRMMesh * mesh;
builder->CreateMesh( &mesh );
// meshにフェースを設定
builder->AddMesh( mesh );
RELEASE( mesh );
```

ではIDirect3DRMMeshにメッシュを登録する方法だが，これにはAddGroup()，SetVertices()，SetGroupColorRGB()メソッドを使う。

```
HRESULT AddGroup(
    unsigned vCount,// 頂点数
    unsigned fCount,// フェース数
    unsigned vPerFace,// フェースの頂点数
    unsigned * fData,// フェースデータ
    D3DRMGROUPINDEX * returnId// グループID
);

HRESULT SetVertices(
    D3DRMGROUPINDEX id,// グループID
    unsigned index,// 開始インデックス
    unsigned count,// 設定する頂点数
    D3DRMVERTEX * values// 頂点データ
);

HRESULT SetGroupColorRGB(
    D3DRMGROUPINDEX id,// グループID
    D3DVALUE red,// 赤要素
    D3DVALUE green,// 緑要素
    D3DVALUE blue// 青要素
);
```

AddGroup()のvPerFaceは，すべてのフェースが同じ頂点数ならばその値を設定するが，異なる頂点数のフェースの場合は0を指定し，フェースデータ側でそれぞれのフェースの頂点数を指定することになる。その場合のフェースデータは，

頂点数
頂点インデックス

の順である。もちろん頂点インデックスは頂点数だけ存在する。これをフェ

リスト1

```
ROKU4
  -0.317041460
   0.094626285
   0.000000000
   0.072112676
   0.022535211
   0.705268893
   1.000000000
   1.000000000
 1
 0
 887.500000000
 887.500000000
 412.000000000
 355.000000000
POINT
 1
 0
 0
 0
 2
  -0.468353748
   0.246354492
   0.326447463
 2
 0
 0
 0
 1
   0.468353748
   0.246354492
   0.326447463
 3
 0
 0
 0
 3
   0.000000000
  -0.281466975
   0.111363997
 4
 0
 0
 0
 4
   0.000000000
   0.169514224
  -0.482517512
LINE0
 1
 3
 0
 0
 2
 1
 0
 0
 1
 4
 0
 0
 4
 2
 0
 0
 2
 3
 0
 0
3
4
0
0
FACE0
3
11
1
6
4
5
3
2
2
6
3
1
3
5
3
4
2
3
3
0
4
5
1
2
END00
PALC0
13
0
233
233
233
105
105
105
0
144
255
255
16
127
127
0
144
144
255
16
16
127
0
255
174
174
127
46
46
0
144
255
144
16
127
16
0
255
255
91
127
127
0
0
255
232
217
192
96
48
0
198
93
53
38
11
0
0
255
255
255
255
255
255
0
0
0
0
0
0
0
0
231
187
150
231
187
150
0
255
18
18
192
0
0
0
248
248
248
185
185
185
0
ENDP0
BAKC0
7
8
255
255
255
0
13
206
212
0
3
4
206
0
255
19
2
0
21
111
16
0
121
78
2
0
0
0
0
0
ENDB0
LIT00
3
0
  0.574988908
  0.511101252
  0.638876565
0
 -0.574988908
  0.511101252
  0.638876565
0
  0.000000000
 -0.668964732
  0.743294146
0
ENDL0
VIEW0
1
1
0
0
ENDV0
LINE2
1
0
9
1
2
0
9
1
3
0
9
1
4
0
9
1
5
0
9
1
6
0
9
1
```

リスト2

```
typedef struct _ROKPOINT {                  // 頂点構造体
                int no;                     // 頂点番号
                int group;                  // グループ
                int show;                   // 表示状態
                int reserved;
                int dim;                    // 対応する対称頂点
                double x, y, z;             // 頂点座標
                // 以降作業用ワーク
                int d3dno;                  // Direct3DRMMesh側対応頂点ナンバー
} ROKPOINT;

typedef struct _ROKLINE {                   // 稜線構造体
                int point[2];               // 稜線を構成する頂点番号
                int group;                  // グループ
                int show;                   // 表示状態
} ROKLINE;

typedef struct _ROKFACE {                   // 面構造体
                int num;                    // 稜線（頂点）数
                int pal;                    // 色番号
                int no;                     // 面番号
                int line[4];                // 稜線番号（六角の面は三角ないしは四角）
                // 以降作業用ワーク
                int point[4];               // 頂点番号
} ROKFACE;

typedef struct _ROKPAL {                    // パレット構造体
                int lred;                   // 明るい所の赤要素
                int lgreen;                 // 明るい所の緑要素
                int lblue;                  // 明るい所の青要素
                int dred;                   // 暗い所の赤要素
                int dgreen;                 // 暗い所の緑要素
                int dblue;                  // 暗い所の青要素
                int reserved;
} ROKPAL;
```

ース数分繰り返し格納する必要がある。

と，ここでちょっと疑問を感じるかもしれない。頂点インデックスといったって，まだ頂点を設定していない。普通と逆である。あくまでもAddGroup()は，グループを作ってメモリを確保するためのメソッドなのだろう。頂点を格納するサイズは頂点数からわかるが，フェースデータは各フェースの頂点数によって可変長であるから，先に指定させようというわけだ。

で，頂点を設定するにはSetVertices()メソッドを使うわけだ。つまり，AddGroup()とSetVertices()はセットで使われることになる。SetVertices()はindexとcount値の制御により，頂点の座標を部分的にセットもできるようになっているが，まとめてセットしたければ，indexは0，countにはそのグループの総頂点数を指定し，valuesには頂点数分のデータを格納すればよい。最後にSetGroupColorRGB()メソッドで，そのグループに含まれているフェースの色を設定する。

なお，フェースが指定できる頂点はそのグループ内の頂点に限られるので，色の違うフェース同士は頂点を(内部データ的に)共有することができない。これはつまり，互い違いに異なる色を指定したオブジェクトの場合，頂点データが膨れてしまうことになる。ま，あんまりむやみにいろんな色をつけないほうがいいということだ。

ところで，ROKはフェースを稜線で記述していると先ほど述べた。Direct3Dはというと，もちろん頂点で管理されている。つまり変換の前に，まずフェースを構成する稜線情報を頂点情報に変換しておくと便利である。このとき，稜線の向きもまちまちであるので気をつけなければならない。順に稜線内の頂点を追っていっただけでは，うまくフェースをぐるりと囲む頂点順にはならないということである。このあたりも踏まえた変換部分をリスト1およびリスト2に示す。すでにROKファイルをそれぞれの構造体配列に読み込まれているものとする。

とりあえずここまでのサンプルがrok2xである。Windowsアプリケーションではあるが，ユーザーインタフェイスは特に設けず(面倒臭かったから)，開いたプレーンなウィンドウにROKファイルをドロップするだけだ。同じフォルダに拡張子がXのファイルが生成される。で，もちろんこのままでは表裏がぐちゃぐちゃである(図2・3)。なんとかせねばなるまい。というか，なんとかするために，とりあえずROKからの変換をテストしたまでだ。

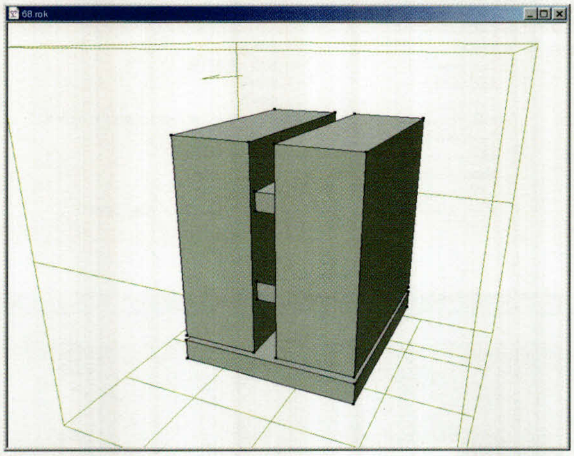


図2

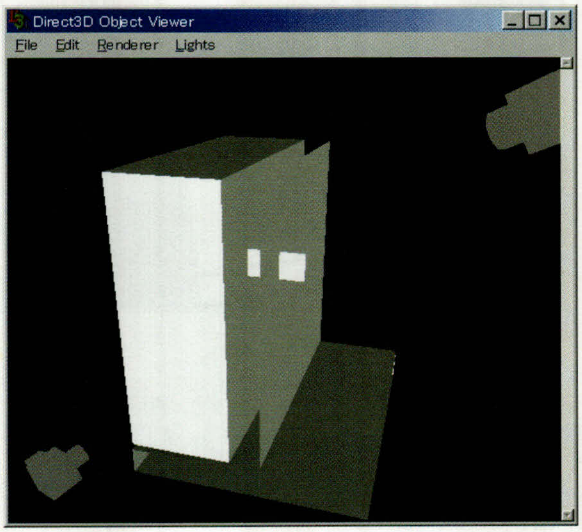


図3

フェース表裏の最適化

さて，どうするか。1枚1枚フェースを表示し，どちらが表かユーザーに問い合わせる。確実だが，それは面倒臭すぎる。面数が少なければいいが，100や200になってくると結構しんどい。要は外側がわかればいいわけだから，3次元のペイントルーチンでも作るか？　とか考えながら，横になって目を閉じ，思考回路を深層に沈めてみた(人はそれを睡眠という)。で，思いついた。高校のときから私は睡眠学習が得意だったのである。レム睡眠さまさまである。

ここに1枚のフェースAがある(図4)。このAは上向きが表であることがわかっているとする。つまり頂点の向きはa→b→cとなる。ではこのフェースと辺abを共有するフェースBはどちらが表だろう。Aとは逆にb→aの順で頂点を構成する向きが表である。つまり頂点の向きはb→a→d(あるいはa→d→b，d→b→a)となる。ではBと辺adを共有するCの向きは……と連鎖して，繋がっている面すべての表がどちらかがわかることになる。

ただし例外もある。飛行機の翼のように，薄い形状を1枚のフェースで作った場合だ。そのフェースは両面描かれることが期待されるが，表も裏もなくなってしまうので，それ以上は連鎖させない。こういった両面フェースを判別する方法も難しくはない。まず，すべての辺に対して，その辺と共有するほかのフェースが存在しなければ，間違いなく両面である。また，特定の辺と共有するフェースがあっても，そのすべてが両面であれば，自分も両面となる。感じとしては，触手を伸ばしていちばん端のフェースを探しているようなものだろうか。端が見つかったら，呼び出し元に遡って端に到達した

リスト3

```
typedef struct _ROKFACE {                    // 面構造体
                    int num;                 // 稜線（頂点）数
                    int pal;                 // 色番号
                    int no;                  // 面番号
                    int line[4];             // 稜線番号（六角の面は三角ないしは四角）
                    // 以降作業用ワーク
                    int point[4];            // 頂点番号
                    BOOL doubleside;         // 両面
                    BOOL proceeded;          // 処理済
                    BOOL nofirstface;// 初期フェースとしては使えない（Y軸に平行）
} ROKFACE;
```

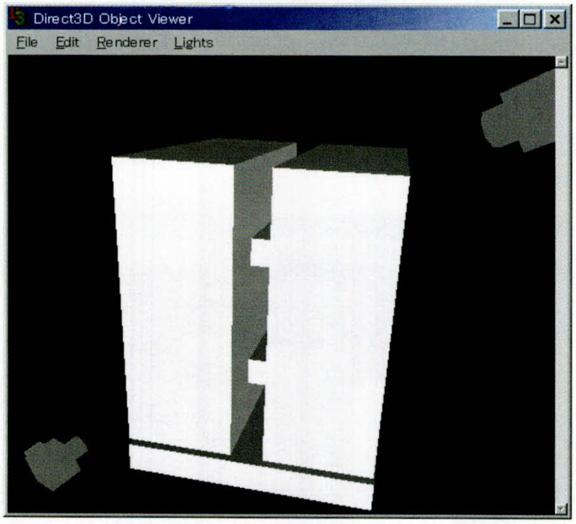


図5

こと（自分たちが両面であること）を通知する。つまり，再帰処理となる。

では最初の大前提，１枚目の表の判別を考えよう。まずはY軸の正方向（別に負方向でもX軸でもZ軸でも構わないのだが）にもっとも遠い頂点を持つフェースを探す。おそらくその頂点は複数のフェースで共有されている（というか，されていなければ両面である）ので，そのうちもっとも外側のフェースを探す。外側の判定は，フェースの重心から共通頂点へのベクトルと，Y軸とのなす角が大きいほうということで調べることができる。そうして検索したフェースは，紛れもなくY軸正向きでもっとも端のフェースといえる。つまり，そのフェースはY方向が表である。

基本はこの2点である。仮に両面フェースにより連鎖が切れてしまったり，元々ひとつのオブジェクトが複数の形状に分かれていて，一度の検索ではすべての表裏の判定ができなくても，残りのフェースに関してまた同じことを繰り返せばよい。ちょっとおさらいしてみよう。

1) Y軸にいちばん遠いフェースを探し，それを起点とする
2) 各辺を共有するほかのフェースを検索し，もし１カ所でも欠けていれば，このフェースは両面であると判定し，呼び出し元に「私は両面です」と申告しに帰る
3) 辺を共有するフェースを再び検索，表裏を最適化してそちらに制御を移し，2)へ再帰
4) 戻ってきた申告をもとに，自分が両面かどうかを判定する。判定方法は，
・特定の１辺について
それを共有するフェースすべてが両面であれば，自分も両面。それ以外は保留
・すべての辺について
１辺でも両面判定が出た場合は，自分は両面。それ以外は片面
その判決結果を持って，呼び出し元に帰る
5) まだ判定していないフェースが残っていた場合は，残り分に関して1)から繰り返し

とまあ，実際はもうちょっと細かい判定が必要だが，これだけなんである。気がついてみるとなんてことはない，コロンブスの卵なんである。これくらいのこと，もうすでに誰かが考えついているかもしれないし，その筋では当たり前のアルゴリズムだったりするかもしれないが，一応自分で考えたと念を押しておいてみようか。あるいは「場」が私に知恵を授けたのかもしれない（なにいってんだか）。

といったことで，その辺の部分をリスト4およびリスト5に示す。少々ごちゃごちゃしているが，私にしてはコメントをちゃんと書いているほうだ。先ほどとは違い，ちゃんとフェースが外側を向くようになっている（図5）。

このアルゴリズムは，もちろんROKからXだけでなく，ほかのコンバータやモデラでも利用できるだろう。活用してもらいたい。

また，再帰しているので，フェース数が多いオブジェクトではスタックフローが気になるところだが，VisualC++のリンカは標準で1MBもの広大なスタック領域を取っているらしい。試しに3000～4000ポリゴンからなるオブジェクトをコンバートしてみたが，フローは起きなかった。実用上必要にして十分なサイズといえる。どうしても気になる人は，2MBでも4MBでも好きなだけ確保すればよい。スタック量は，［プロジェクト］［設定］で開くダイアログから［リンク］タブをクリックし，［カテゴリ］で［アウトプット］を選択後，［スタックアロケーション］［予約］テキストボックスにバイト単位で指定できる。ただし，コンバート中はメッセージの表示などは一切なく，メッセージも受け取らないので，巨大なオブジェクトの変換中は，結構不安になったりするかもしれない。それでも3000～4000ポリゴン程度でCeleron 450Aなら10秒程度だったので，普通に速いマシンなら何分も反応しないといったことは（正常に動いている限り）ないはずだ。

注意点がいくつか。まずひとつは外のフェースと辺を共有する内部のフェースを作らないこと。言葉ではわかりにくいが，図6のような場合だ。立方体のそれぞれの周囲面を三角形分割したフェースが張られているとする。

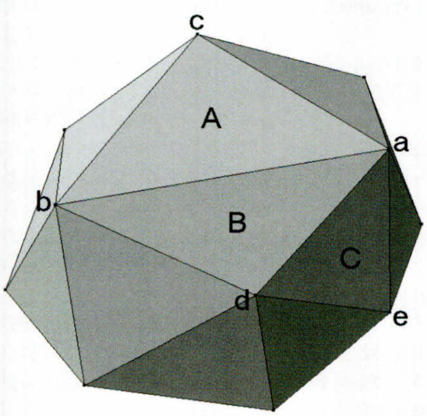


図4

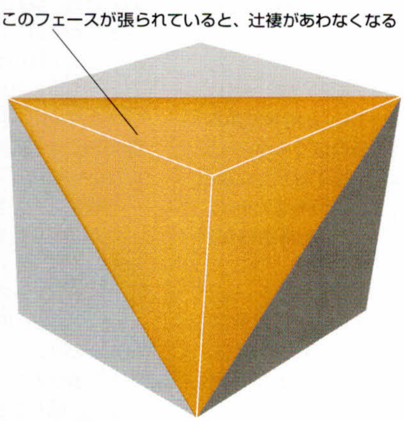


図6　正常に処理できない例

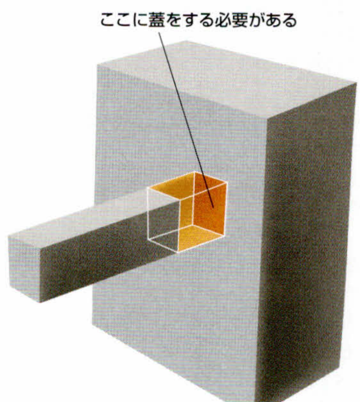


図7　独立したオブジェクトでは空間を閉じられない

リスト5

```
void OptimizeFace()
{
  for( int i=0; i<facenum; i++ )
    RokFace[i].doubleside = RokFace[i].proceeded = RokFace[i].nofirstface = FALSE;
  BOOL b;
  do {
    int n;
    n = SearchFirstFace();
    if( n>=0 ) ChainFace( n );
    b = TRUE;
    for( i=0; i<facenum; i++ ) b &= RokFace[i].proceeded|RokFace[i].nofirstface;
  } while( !b );
  // Y軸に水平で処理していないフェースが残ってるかも。そりゃ両面だろうな
  for( i=0; i<facenum; i++ )
    if( RokFace[i].nofirstface )
      if( !RokFace[i].proceeded ) RokFace[i].doubleside = TRUE;
}

int SearchFirstFace()
{
  int n, m;
  double y=-10000.0;
  D3DVECTOR v1, v2, vr;
  for( int i=0; i<facenum; i++ ){
    if( !RokFace[i].proceeded && !RokFace[i].nofirstface ){
      for( int j=0; j<RokFace[i].num; j++ ){
        // Y軸の正方向に一番遠いフェースを探す
        if( y<RokPoint[RokFace[i].point[j]].y ){
          y = RokPoint[RokFace[i].point[j]].y;
          n = i;
          m = j;
        } else if( y==RokPoint[RokFace[i].point[j]].y ){
          // 重心から共通頂点を見て、Y軸との角度が深い方のフェースを選ぶ
          v1.x = RokPoint[RokFace[n].point[m]].x*2
            -RokPoint[RokFace[n].point[m<RokFace[n].num-1?m+1:0]].x
            -RokPoint[RokFace[n].point[m>0?m-1:RokFace[n].num-1]].x;
          v1.y = RokPoint[RokFace[n].point[m]].y*2
            -RokPoint[RokFace[n].point[m<RokFace[n].num-1?m+1:0]].y
            -RokPoint[RokFace[n].point[m>0?m-1:RokFace[n].num-1]].y;
          v1.z = RokPoint[RokFace[n].point[m]].z*2
            -RokPoint[RokFace[n].point[m<RokFace[n].num-1?m+1:0]].z
            -RokPoint[RokFace[n].point[m>0?m-1:RokFace[n].num-1]].z;
          D3DRMVectorNormalize( &v1 );
          // Y軸とFace[n]の成す角
          v2.x = RokPoint[RokFace[i].point[j]].x*2
            -RokPoint[RokFace[i].point[j<RokFace[i].num-1?j+1:0]].x
            -RokPoint[RokFace[i].point[j>0?j-1:RokFace[i].num-1]].x;
          v2.y = RokPoint[RokFace[i].point[j]].y*2
            -RokPoint[RokFace[i].point[j<RokFace[i].num-1?j+1:0]].y
            -RokPoint[RokFace[i].point[j>0?j-1:RokFace[i].num-1]].y;
          v2.z = RokPoint[RokFace[i].point[j]].z*2
            -RokPoint[RokFace[i].point[j<RokFace[i].num-1?j+1:0]].z
            -RokPoint[RokFace[i].point[j>0?j-1:RokFace[i].num-1]].z;
          D3DRMVectorNormalize( &v2 );
          if( v2.y<v1.y ){
            y = RokPoint[RokFace[i].point[j]].y;
            n = i;
            m = j;
          }
        }
      }
    }
  }
  v1.x = RokPoint[RokFace[n].point[1]].x-RokPoint[RokFace[n].point[0]].x;
  v1.y = RokPoint[RokFace[n].point[1]].y-RokPoint[RokFace[n].point[0]].y;
  v1.z = RokPoint[RokFace[n].point[1]].z-RokPoint[RokFace[n].point[0]].z;
  D3DRMVectorNormalize( &v1 );
  v2.x = RokPoint[RokFace[n].point[RokFace[n].num-1]].x-RokPoint[RokFace[n].point[0]].x;
  v2.y = RokPoint[RokFace[n].point[RokFace[n].num-1]].y-RokPoint[RokFace[n].point[0]].y;
  v2.z = RokPoint[RokFace[n].point[RokFace[n].num-1]].z-RokPoint[RokFace[n].point[0]].z;
  D3DRMVectorNormalize( &v2 );
  // Y軸の正方向を向いている方が表
  D3DRMVectorCrossProduct( &vr, &v2, &v1 );
  if( vr.y<0.0 ){ // 裏返す
    ReverseFace( n );
  } else if( vr.y==0.0 ){ // せっかく探したのに、Y軸に平行だねこりゃ。使えないよ
    RokFace[n].nofirstface = TRUE;
    n = -1;
  }
  return n;
}

void ReverseFace( int n )
{
  int m = RokFace[n].num;
  for( int i=0; i<m/2; i++ ){
    int tmp = RokFace[n].point[i];
    RokFace[n].point[i] = RokFace[n].point[m-1-i];
    RokFace[n].point[m-1-i] = tmp;
  }
}

// チェーンが続く場合（片面の場合）TRUEを返す
BOOL ChainFace( int n )
{
  RokFace[n].proceeded = TRUE;
  BOOL result = TRUE; // 面を構成する稜線のうち、一箇所でもチェーンしない部分があればチェーン終了
  for( int i=0; i<RokFace[n].num; i++ ){
    // とりあえずチェック。1稜線でも他のフェースと共有していなければ両面
     result &= SearchFace( n, RokFace[n].point[i], RokFace[n].point[i>0?i-
1:RokFace[n].num-1], FALSE );
    if( !result ){
      RokFace[n].doubleside = TRUE;
      return result;
    }
  }
  result = TRUE;
  for( i=0; i<RokFace[n].num; i++ ){
    // 本格的にチェーンして表裏を決定
     result &= SearchFace( n, RokFace[n].point[i], RokFace[n].point[i>0?i-
1:RokFace[n].num-1], TRUE );
    if( !result ){
      RokFace[n].doubleside = TRUE;
      return result;
    }
  }
  return result;
}

// チェーンが続く場合（片面の場合）TRUEを返す
BOOL SearchFace( int n, int n1, int n2, BOOL b )
{
  BOOL result = FALSE;  // この稜線に繋がるフェースのうち、一つでもチェーンすれば続行
  for( int i=0; i<facenum; i++ ){
    if( i==n ) continue;
    for( int j=0; j<RokFace[i].num; j++ ){
      if( RokFace[i].point[j]==n1 ){
        if( RokFace[i].point[j<RokFace[i].num-1?j+1:0]==n2 ){
          // 表向きで稜線を共有しているフェース
          if( !RokFace[i].proceeded ){
            if( !b ) return TRUE;
            result |= ChainFace( i );
          } else {
            // 隣がすでに処理したフェースでも、それが片面ならたぶん大丈夫
            if( !b ){
              if( !RokFace[i].doubleside ) return TRUE;
            } else result |= !RokFace[i].doubleside;
          }
        } else if( RokFace[i].point[j>0?j-1:RokFace[i].num-1]==n2 ){
          // 裏向きで稜線を共有しているフェース
          if( !RokFace[i].proceeded ){
            if( !b ) return TRUE;
            ReverseFace( i );
            result |= ChainFace( i );
          } else {
            if( !b ){
              if( !RokFace[i].doubleside ) return TRUE;
            } else result |= !RokFace[i].doubleside;
          }
        }
      }
    }
  }
  return result;
}
```

こういった形状を作った場合，六角大王で［面塗り］［全ての面を塗る］コマンドでは，内部に外からでは見えない面が張られてしまう。これがあると，rok2xは表裏の辻褄があわなくなり，正常に最適化できない。ROKの出力の際には，必ず［編集］［稜線を選択］［3面稜線を選択］してこういったフェースを洗い出し，［面塗り］［面削除］で削除すること。

もう一点。稜線（頂点）を共有していないフェース同士では，空間を閉じることはできない。これはアルゴリズムを考えるというまでもないことだが，図7のような場合，奥の端面も張ってやらないことには，棒の内部にも面が張られてしまう。これが四角い棒ならばフェースが1枚増えるだけで済むが，複雑な形状ならば馬鹿にならない。そこで，真っ黒いフェースは表裏判定のときだけ使用され，出力時には削除するオプションをつけた（図8）。黒いフェースで蓋をしておき，コンバート時にこのオプションをチェックしておけば，無駄なフェースを出力することなく，棒の内側のポリゴンを省略できるというわけだ。

そんなわけで，これでなんの気兼ねもなく六角大王でモデリングできるようになったわけだ。さて，特集のほうのキャラクターのデザインするかな。

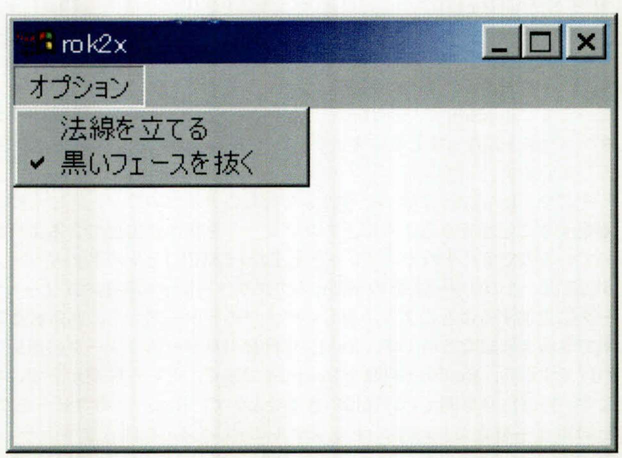


図8

FutureBASICを使った AfterDarkモジュールの作り方

水野 貴明　Mizuno Takaaki

画面を彩るスクリーンセーバー。AfterDarkのモジュールはFutureBASICでも開発可能です。ここではAfterDarkのモジュールSDKを使って実際に独自のモジュールを作ってみましょう。

空飛ぶトースターで有名なAfterDarkは，Berkeley Systems社が販売しているスクリーンセーバーソフトウェアで，モジュールと呼ばれるファイルを追加することにより，さまざまな種類のスクリーンセーバーを楽しめるという優れものです。Windowsのように標準でスクリーンセーバー機能を持たないMacintoshにおいては定番ともいえるソフトウェアであり，「BeforeDark」「DarkSide of the Macintosh」といった，AfterDarkの名前をもじったオンラインソフトウェアもあるほどです。今回はFuture BASIC IIを使って，AfterDarkのモジュールを作る方法について簡単に説明していきたいと思います。

準備

AfterDarkは仕様も公開されていますので自分でモジュールを作ることができます。新しいモジュールを開発するために，Berkeley Systems社はSDK (Software Development Kit) を公開しています。

・After Dark 3.0 SDK
http://www.berksys.com/ftp/support/mac/afterdark/developer/ad30sdk.sit.hqx

とりあえず，SDKを落としてきましょう。このなかには，AfterDarkモジュールを作るための詳しい資料やC言語およびPascalのサンプルコード，ResEditのテンプレートファイルなどが入っています。お気づきかと思いますが，AfterDarkの現バージョンはver.4.Xなのに，SDKはver.3.0のものです。タイムスタンプを見ても1995年になっています。したがって，4.0以降で実装された機能があったとしても使い方がわかりませんが，ver.4のSDKを発見することができませんでした（はたして，ver.4.XのSDKは存在しているのでしょうか？　知っている方は教えてください）。とはいえ，ver.3の機能だけを使ってもアイデア次第で十分いろいろなものが作れますので，大丈夫です。

さて，まずResEditにテンプレートを追加しなくてはいけません。AfterDarkモジュールは，コントロールパネル用のスライダー，チェックボックスといったアイテムの管理や，そのほか挙動を制御するためにいくつものリソースを用います。そのリソースはありふれたタイプのものもありますが，AfterDark用に新しく定義された構造のものもあります。そういったいままでResEditが「知らない」リソースをResEditで開くと，16進数のダンプリストになってしまいますが，これでは編集がものすごく大変になってしまいます。

そこで，ResEditには，そういった新しいタイプのリソースでも簡単に編集することができるように，テンプレートを新たに追加できるようになっているのです。そのテンプレートもまた「TMPL」という名のリソースです（図1）。このリソースをResEditのアプリケーション本体のリソースフォークに追加することにより，新しいタイプのリソースでも，ある程度は編集できるようになるのです。ResEdit自身のリソースフォークに追加するわけですから，ResEdit自身をResEditで開く，という作業が必要になります。自分自身を開くわけにはいきませんので，ResEditのコピーを作り，そのコピーを立ち上げて，オリジナルのResEditを開きます。そして，SDKについてきたリソースファイル（After Dark ResEdit TMPLs）を開き，TMPLリソースを丸ごと，オリジナルのResEditのリソースフォークにコピーします。

ResEditは標準でいくつかのTMPLリソースを持っていますが，それらのリソースと，追加するリソースは，IDが重ならないようになっていますので，そのまますんなりコピーできるでしょう。もし，なんらかのほかのテンプレートを追加している場合には，「IDがぶつかっている」という警告が出ますが，そのダイアログで「Unique ID」を指定して，空いているIDに割り当ててやれば問題ないです。作業が終わったら，間違えないように，コピーしたResEditは消しておきましょう。

さて，これで，ResEditの準備ができました。この，テンプレートをコピーしたResEditを使えば，「sVal」「mVal」といったAfterDark専用のリソースの作成，編集を容易に行うことができます。

コードリソースを作る

AfterDarkのモジュールは，コードリソースと呼ばれるタイプのプログラムです。コードリソースとは「メモリに読み込まれて実行されるリソース」のことで，プログラムをリソースデータとして持っておき，好きなときにメモリにロードして呼び出すことができます。これは，AfterDarkのモジュールやPhotoshopなどのプラグイン，HyperCardのXCMDなどのように，追加機能を別ファイルとして持っておきたいときに大変役立ちます。なかなか便利なコードリソースですが，普通のアプリケーションに比べ，若干の制限があります。それは，ひとつのセグメントしか使えないことです。普通のアプリケーションなら，プログラムサイズが大きくなったときには，SEGMENT文を使ってセグメントをいくつも分割することが可能でしたが，コードリソースではそれができません。それ以上のサイズのものを作る場合には，コードリソースそのものを分割する必要があります。また，サイズを小さくまとめるために，通常のランタイムパッケージの代わりに，ミニランタイムパッケージが使われます。このランタイムパッケージはサイズがすごく小さく抑えられているのですが，その分，使える命令数も少なくなっています。どの命令が使え，どの命令が使えないかについては，リファレンスマニュアルを参考にしてください。命令の名前の横に，リソースファイルのマークが書かれているものが，コードリソースで使用できる命令です。

コードリソースは，そのコードリソースを呼び出すアプリケーション（もしくはコードリソース）とのやりとりに，ENTERPROC%～EXITPROC%（*1）というものを使います。これは，LOCAL FN～END FNと似たよう

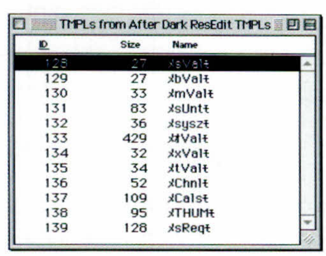


図1　AfterDarkモジュール用のリソーステンプレート

な感覚で使うものです。アプリケーションからコードリソースが呼び出されると，ENTERPROC%～EXITPROC%で囲まれた部分が実行されるのです。ENTERPROC%～EXITPROC%はアプリケーションからパラメータを受け取ることもできますし，戻り値を返すこともできます。しかし，注意しなくてはいけないのは，呼び出すアプリケーションは特定できませんから，コンパイルするときに，パラメータ，戻り値の数や順番，型が間違っていても，FutureBASIC IIはエラーだとは気づきにくい点です。間違った指定をしたままビルドして，実行すると，ひどいときにはシステムエラーになってしまいますから，注意が必要です。

さて，AfterDarkの既存のモジュールをResEditで開いてみると，いくつものリソースが含まれていることがわかります(図2)。「PICT」や「TEXT」といったお馴染みのものもありますが，「ADgm」「sVal」「mVal」といった見慣れないものもあると思います。これらはAfterDark専用のリソースです。そのなかで「ADgm」というのが，モジュール本体のコードリソースになります。そのほかは，モジュールの制御に使うためのリソースです。

AfterDarkモジュールのしくみ

AfterDarkのモジュールのすべき仕事は，スクリーンセーバー画面の描画とコントロール，およびコントロールパネルでの設定に対する反応，の2つです。モジュールが処理をすべきときには，AfterDark本体がモジュールを呼び出します。その際に，モジュールがなにをすべきかということは渡されるパラメータのなかで指定されていますので，それに応じて処理を行えばいいのです。

モジュールに渡されるパラメータ

AfterDarkがモジュールに渡してくるパラメータを表1に示しました。全部で4つです。

storage&

これは，AfterDarkモジュールが内部で使用するグローバル変数を記憶しておくためのメモリのハンドルです。通常は，モジュールの初期化の際に，モジュールの実行に必要なメモリサイズのハンドルを取得し，モジュール実行中はそのハンドルにパラメータを保存しておき，モジュールの終了時に破棄する，という使い方をします。が，FutureBASIC IIでコードリソースをコンパイルするときには，COMPILE文で_sysHeapVars_pointerVarsを指定してやればFutureBASIC IIが勝手にグローバル変数を処理してくれるようなので，このハンドル内に変数を記憶する必要はないのです。しかし，AfterDark本体に処理を返したとき，このハンドルがNULLだと，AfterDarkはモジュールの起動に失敗したと思ってしまうようなので，一応，初めにハンドルを取得する必要があります。

blankRgn&

これは，モジュールが描画できる領域を示す領域ハンドルです。モジュールから，このなかへ描画することにより，ディスプレイを焼き付きから防ぐわけです。通常は画面いっぱいの領域となっていますが，デモモード(コントロールパネルの「デモ」ボタンを押してモジュールを実行している状態)の場合は，コントロールパネルが表示されたままですので，コントロールパネルの部分が領域からはずれます。

message%

これは，モジュールが呼ばれた際に，なにをするべきかをAfterDarkが伝えるためのもので，普通のプログラムのイベントに当たるようなものです。その種類を表2に示します。

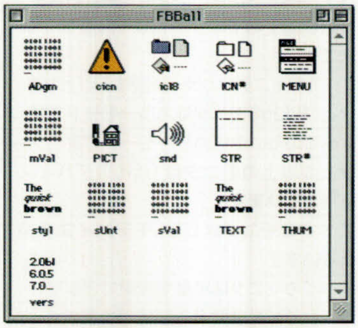


図2　AfterDarkモジュールの持つリソース

表1　AfterDarkがモジュールに渡すパラメータ

順番	パラメータ
1	storage&
2	blankRgn&
3	message%
4	params&

表2　AfterDarkがモジュールに送ってくるメッセージ

番号	メッセージ名	説明
0	Initialize	モジュールが開始するときに1回だけ送られてくるメッセージです。デモモードでは，デモボタンが押されるたびにいちばん初めに送られてきます。これが送られてきたら，必要なだけのメモリを割り当てたハンドル(storage&)を作る必要があります。モジュール内で使う変数の初期化などもここで行います。ただし，ここではまだ画面の描画を行ってはいけません
1	Close	モジュールが終了するときに送られてくるメッセージです。ハンドル(storage&)の破棄を行います。画面はAfterDarkによって元の状態に戻されるので，ハンドルの破棄以外は，なにもする必要はありません
2	Blank	Initializeメッセージの次に送られるメッセージです。ここで画面の初期描画を行います。たとえば，画面を真っ暗にする，絵を表示する，などです
3	DrawFrame	モジュールが実行されている限り，一定期間ごとに何回も送られてくるメッセージです。Timer EVENTみたいなものです。ここで，アニメーションなどを行います
4	ModuleSelected	このメッセージは，コントロールパネルでそのモジュールが選択されたときに送られてきます。たとえば，モジュールが選択されたときに，現在のフォントメニューを表示したい，スライドバーの状態を初期設定に戻したい，といったときに使います。このメッセージは，Calsリソースを作り，メッセージ発行を要求しない限り，送られてくることはありません
5	DoHelp	このメッセージはコントロールパネルのクレジットエリアがクリックされ，ヘルプボックスが表示されるときに送られてきます。ただし，ヘルプボックスにピクチャ，もしくはテキストをただ表示するだけなら，プログラムを組まなくてもリソースを追加するだけで実現できます。このメッセージも，ModuleSelectedメッセージと同じく，alsリソースを作り，メッセージ発行を要求しない限り，送られてくることはありません
8～	ButtonMessage	コントロールパネルにボタンをつけていた場合で，そのボタンが押されたときにこのメッセージが送られてきます。その番号は，ボタンを設定するbValリソースで設定する8以上の整数です。ボタンが押されたときに，カラーピッカーを表示する，などといった使い方をします

*1 ENTERPROC～EXITPROCという「%」がないものもありますが，これはコールバックルーチンに使用するもので，目的が違うので注意してください。

params&

コントロールパネルで設定した各種値や，モニタの色深度，AfterDarkのバージョンなどの，モジュールの実行の際に必要，もしくは簡単にわかると非常に便利なデータを格納したレコードへのポインタです。このレコードは，さらに別のレコードへのポインタも内包している，若干複雑な構造になっています。AfterDarkモジュールで使うすべてのレコードの定義リストをリスト1に示します。リストのいちばん下の，GMParamsBlockRecが，params&の構造です。

controlValuesはコントロールパネルの各種設定用のスライダーの値や選択されたメニュー，チェックボックスの現在の状態などが入っているレコードへのポインタです。その構造はctrlArrayにitem1%からitem4%と定義されているわけですが，4つしかありません。なぜ4つしかないかというと，AfterDarkのコントロールパネルはそういったアイテムを4つまでしか持てないからです。item1%〜item4%の値は，スライダーなら1から100の値が，メニューなら選択されたメニューの番号が，テキストなら定義されたテキストの番号，チェックボックスなら0か1が入っています。

monitors&は，MonitorsInfoで定義されたレコードへのポインタです。そこで定義されているのは，Macintoshに接続されているモニタの数を示すmonitorCount%と，それぞれのモニタの情報の入った，monitorDataで定義された構造を持つmonitorListです。monitorDataには，boundsRect（SDKでは，boundsと定義されていますが，この名前はFutureBASIC IIでは既存の変数ですでに使われている名前ですので，名前を変えています）は，そのモニタの矩形が入ります。synchFlag%はモニタの垂直同期のフラグで，よりスムーズなアニメーションを実現するのに使用します。curDepth%にはそのモニタのカラー深度が入ります。

colorQDAvail%は，カラーQuickDrawがそのMacintoshに実装されているかを示しています。これは純粋にカラーQuickDraw用のライブラリが実装されているかを示し，そのMacintoshがカラーモニタにつながっているか，もしくは現在カラーモードになっているかを示すものではないので注意が必要です。

systemConfig%は，16ビットのフラグのセットです。その構造を表3に示します。systemという名前がついていますが，示されている情報はそんなに多くはありませんので，もしもっと詳しくシステム情報を知りたいと思ったら，SysEnvironsやGestaltなどのToolBoxルーチンを用いて情報を持ってくる必要があります。

qdGlobalsCopy&はQuickDrawのグローバル変数を参照できるものですが，あくまでコピーですので，この変数を変更したからといって反映されません。その構造はqdGlobalsに定義されています。

brightness%は，変更することによってモニタの輝度を設定できます。値は0から255の間で指定します。ただしsystemConfig%のallCanDimもしくはmainMonCanDimのビットが1になっていないと，変更は反映されません。

demoRectは，「デモモード」でのみ意味のある値です。コントロールパネルの「デモ」ボタンを押すことで，デモモードになるわけですが，その場合，コントロールパネルが画面上に残ります。この変数にその矩形が入っています。また，デモモード以外では，この変数には空の矩形が入りますので，現在の状態がデモモードなのかどうかを判別するのにも使えます。

errorMessage&にはエラーメッセージを格納しておく領域へのポインタが入っています。AfterDarkでは，なにか問題が起こると画面のいちばん上，ちょうどメニューバーがあるあたりにメッセージが流れます。あのメッセージを格納しておくための場所です。255字まで格納できます。

sndChannel&は，サウンドを鳴らすときに必要なサウンドチャンネルが入っています。Chnlリソースをモジュールに追加することにより，AfterDarkはサウンドチャンネルを割り当てて，そのポインタをこの変数に入れてくれます。

adVersion%にはAfterDarkのバージョンが入っていますが，BCDという形式で入っていて，要するに16ビットを4ビットごとに分けてそれぞれに0から9までの値を入れてあるというものです。たとえばバージョン2.00なら0x0200という値（10進数なら512）が入ります。バージョン4.04なら0x0404となります。

表3　systemConfig%の各ビットの意味

ビット	名称	意味
0	cqdAvail	カラーQuickDrawが実装されています
1	anyMultiBit	ひとつ以上のモニタのカラー深度は1以上です
2	allMultiBit	すべてのモニタのカラー深度は1以上です
3	anyColor	ひとつ以上のモニタはカラーモードです
4	allColor	すべてのモニタはカラーモードです
5	anyCULTDevice	ひとつ以上のモニタはCULTデバイスを持っています
6	allCULTDevice	すべてのモニタはCULTデバイスを持っています
7	allCanDim	すべてのモニタは輝度を変更できます
8	mainMonCanDim	メインモニタは輝度を変更できます
9	moduleMayNotAnimate	モジュールはカラーテーブルをアニメートすべきではない
10	multiModuleRunning	モジュールはマルチモジュールモードで実行されています
11-13	予約済み	
14	extensionsAvailable	モジュールはAfterDarkのバージョン2.0かそれ以上で実行されています
15	soundAvail	このバージョンのAfterDarkはサウンドをサポートしています

モジュールが返す値

モジュールは，AfterDarkからmessage%で指定された作業を終えたあとAfterDarkに処理を返すわけですが，その際に整数値を返します。これは数値ごとに意味づけがされており，モジュールの処理の結果や状況をAfterDarkに知らせるために使います。その数値と意味は，表4のとおりです。

リスト1　AfterDarkモジュールで使うレコードの構造

```
DIM RECORD monitorData
  DIM monitorRect.8
  DIM synchFlag%
  DIM curDepth%
DIM END RECORD.monitorData

DIM RECORD MonitorsInfo
  DIM monitorCount%
  DIM monitorList.monitorData
DIM END RECORD.MonitorsInfo

DIM RECORD ctrlArray
  DIM item1%
  DIM item2%
  DIM item3%
  DIM item4%
DIM END RECORD.ctrlArray

DIM RECORD qdGlobals
  DIM qdThePort&
  DIM qdWhite.8
  DIM qdBlack.8
  DIM qdGray.8
  DIM qdLtGray.8
  DIM qdDkGray.8
  DIM qdArrow.16
  DIM qdScreenBits&
  DIM qdRandSeed&
DIM END RECORD.qdGlobals

DIM RECORD GMParamBlockRec
  DIM controlValues.ctrlArray
  DIM monitors&
  DIM colorQDAvail%
  DIM systemConfig%
  DIM qdGlobalsCopy&
  DIM brightness%
  DIM demoRect.8
  DIM errorMessage&
  DIM sndChannel&
  DIM adVersion%
DIM END RECORD.GMParamBlockRec
```

表4　モジュールがAfterDarkに返す値

値	名称	意味
0	noErr	なにも問題がないときに返す値です
−1	ModuleError	モジュール実行中になにか問題が起こったときに返します。いちばん多いのは，イニシャライズ中にメモリの割り当てに失敗したときなどです。それ以外にもモジュール実行中にエラーが起こったら，このメッセージを返します。そのとき注意しなくてはいけないのは，割り当てたメモリをすべて破棄してからエラーを返さなくてはいけないことです
1	RestartMe	ADはこのメッセージを受け取ると，Closeメッセージを出すことなくモジュールを再起動します（つまりInitializeメッセージを送ってきます）。このメッセージは，デモモードでパラメータが変更されたときなどに使うとよいでしょう。ただし，その際Closeメッセージを受け取ったときと同じ処理をするのをお忘れなく
2	ImDone	このメッセージはAfterDarkに描画を引き継いでもらいたいときに送ります
3	RefreshResource	このメッセージを送ると，コントロールパネルのスライダーやチェックボックスやメニューなどが再描画されます。ButtonMessageとModuleSelectedに対する戻り値として使用されます

モジュール設定用のアイテム

AfterDarkのコントロールパネルは，図3のような作りになっています。ここでモジュールを選択すると，モジュールのさまざまな設定ができるように，図4のようなコントロールインタフェイスが現れます。続いて，これらのアイテム，たとえばスライダーやポップアップメニュー，チェックボックスなどを作る方法を説明します。

これらのアイテムの位置と種類はリソースで定義されています。といっても，リソースのなかに位置や種類が書き込まれているわけではなく，リソースタイプとIDで位置と種類を決定しているのです。

さて，ひとつのモジュールは4つまでしかコントロールアイテムを持つことはできませんので注意してください。まあ，スペース的にも4つが限界なんでしょう。そして，それぞれのアイテムの位置を表すリソースIDは1000から1003です。それぞれのアイテムを表すタイプ，たとえばスライダーの場合はsValというリソースを作って，そのリソースID（複数のタイプのリソースでひとつのアイテムを定義することもあるので，その場合はそれらすべて）を置きたい場所の値にすれば，あとはAfterDarkがきちんとその意味を理解して，コントロールパネルに表示してくれるわけです。

それでは，各アイテムを種類ごとに見てみましょう。

スライダー

スライダーは左右に動かして連続的な値を設定することのできるアイテムです。いろいろなモジュールを見ていても，いちばん使われてるように思います。

スライダーを作るためには，sValというリソースを作ります。sValは，中身は2バイト長とものすごく小さいリソースです。この中身は，スライダーの現在の値が入っていて，範囲は0から100です。モジュールを作るときにはここにデフォルトの値を設定しておきます。この値が，params&のcontrolValuesに入ってくるわけです。ちなみにこのリソースの名前が，コントロールパネルに表示される名前になるので，任意の名前をつけてください。「速度」「出現頻度」とかといった具合です。このことはテキストを除くすべてのアイテムでも同様です。

さて，sValリソースを作っただけでは，スライダーの右側に表示される値は数字です。しかし，モジュールのなかには，そのスライダーの値に応じて，数字ではなくて「おそい」「ふつう」「はやい」といった，状態を表す文字が出てくるようなスライダーもあります。そういったスライダーを作るためには，もうひとつsUntというリソースを作ります。このとき，そのIDは，対応するsValと同じ値にします。sUntリソースは，2バイト長のデータと文字列のセットを複数持てるリソースで，それぞれの文字列が表示される最低値を入れます。たとえば，先ほどの「おそい」「ふつう」「はやい」という文字列を表示させたい場合は，ひとつ目のセットに0と「おそい」，2つ目のセットに34と「ふつう」，3つ目のセットに67と「はやい」を入れておけば，値が0〜33の場合に「おそい」，34〜66のとき「ふつう」，67〜100のときに「はやい」と，3分の1ずつの区切りで表示されるようになります（図5）。

ポップアップメニュー

続いて，ポップアップメニューです。これはいわゆる普通のポップアップメニューと同じなので，特に説明はいらないかと思います。ポップアップメニューを作るためのリソースはmValです。中身はやはり2バイト長で，いま選択されているメニューの上から数えた番号が入ります。1から始まって，そのメニューの選択肢の最大値までの範囲になるわけです。そのメニューの中身を定義するのは通常のメニューと同様に，ENUリソースで行います。通常のメニューを作成するときとまったく同じようにメニューを作成して，IDをmValと同じにすればよいのです。

チェックボックス

チェックボックスを作るリソースはxValです。中身は2バイト長で，入るのは0か1，そのチェックボックスがチェックされていれば1，されていなければ0です。



図3　AfterDarkコントロールパネルの構成

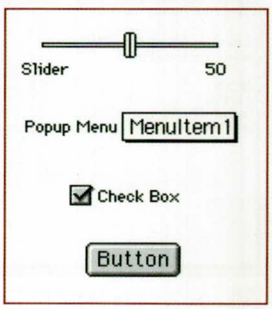


図4　AfterDarkモジュールの設定に使うコントロール

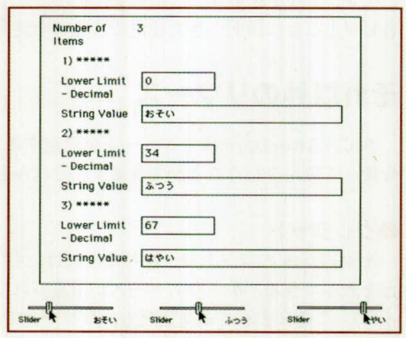


図5　sUntリソースのしくみ

ボタン

お馴染みのボタンです。リソースはbValです。リソース名がボタンに描かれます。ボタンはほかのアイテムと違って，数値を変更したりすることはなく，その代わりにモジュールにメッセージ「ButtonMessage」を送ります。中身は2バイトで，値は8以上の任意の数値です。つまり，8以上ならなんでもかまいません。ただし，複数のボタンがある場合には，ボタン同士で重複しないようにしなくてはいけません。なぜなら，ButtonMessageというメッセージを送るということは，この数値そのものをmessage%に入れて送ることを意味するので，重複しているとモジュールがどのボタンが押されたか特定できないからです。

テキスト

これはテキストをそのままコントロールパネル上に表示するもので，リソースはtValです。このアイテムの場合は唯一例外的に，リソースの名前は画面に表示されません。表示されるのはメッセージのみです。表示されるテキストはSTR#リソースのtValと同じIDに格納されているものです。STR#は，ひとつのIDにいくつもの文字列を格納できるので，tVal内（2バイト）に表示するテキストの番号（1以上）を入れます。tValに0が入っているとなんのテキストも表示されません。モジュール内でアイテムの値を変えることにより，表示される文字列を変化させることができます。

条件づけ

最後に，コントロールそのものではありませんが，コントロールの表示，非表示をコントロールする，ｵValというリソースについて説明します。「ｵ(&hB5)」という文字は，英文フォントでは「μ」となり，Option+mで表示することができます。

このリソースは，やはり2バイト長のリソースですが，その16ビットが，前述したパラメータ，param%のsystemConfig%に対応した，16ビットのフラグセットになっています。このリソースを，対象となるコントロールと同じIDで作成しておくと，systemConfig%の値と，ｵValリソースが比較され，ｵValリソースで1になっている（フラグが立っている）ビットと同じビットが，systemConfig%でも1になっている場合のみ，そのコントロールは表示され，systemConfig%で0になっている場合には，コントロールは表示されません。たとえば，「カラーQuickDrawがある場合のみコントロールを表示したい！」といった場合には，対応するのは第0ビットですので，ｵValリソースの値を1にしておけばいいのです。サウンドが使える場合，というのであれば，対応するのは第15ビットなので，&h8000ということになります。

また，このリソースを使うと，「カラーQuickDrawがある場合はこのコントロール，ない場合はこっちのコントロール」といった，状況によってコントロールを切り替えるようなことも可能です。そのためには，リソースIDが2000～2003のリソースを使います。もし，1000～1003までのIDのコントロールが，ｵValによって表示されなくなったとき，そのリソースIDに1000を加えたIDのリソースが存在すると，そちらを表示してくれるというものです。

テンプレートを追加したResEditを使って，ｵValリソースを編集すれば，各ビットごとに説明つきで設定できるので便利です。

それ以外のリソース

さて，AfterDarkはモジュールを設定するアイテム以外にも，リソースを使って設定を行うことができます。それらを順に見ていきましょう。

●クレジット

AfterDarkのコントロールパネルには，右下に各モジュールの制作者などを表示できる，小さなスペースが用意されています（図3のクレジットライン）。ここに表示する文字列も，リソースで設定できます。使うリソースは「STR 」リソース，IDは128です。いうまでもありませんが，リソース名は4文字ですので，STRの後ろにはスペースがひとつ入っています。

●ヘルプボックス

クレジットが表示されているスペースをクリックすると，コントロールボックス全面を使ったヘルプ画面が表示されます。このヘルプ画面は，テキスト（図6），ピクチャ（図7）のどちらも使うことができます。このヘルプの内容も，やはりリソースで作ります。使うのは，TEXTリソースと，PICTリソースで，IDは1000です。

TEXTリソースとPICTリソースどちらかにIDが1000のリソースを作っておけば，その内容が表示されます。両方作ると，TEXTリソースが優先されるようです。テキストには，スタイルをつけることもできます。ResEditでTEXTリソースにスタイルを設定すれば，自動的に同じIDの「styl」リソースを追加してくれます。なお，表示エリアは260×272です。したがって，このサイズ以上のテキストやピクチャを表示させようとすると，一部切れてしまいますので，注意が必要です。

ヘルプボックスは，リソースIDを1000だけでなく，1001，1002，と複数持つことにより，クリックにより，次々に切り替えていくことができます。1枚目はテキスト，2枚目はピクチャ，3枚目はまたテキスト，と自由に組み合わせることができます。ただし，IDは連続していなくてはならず，抜けがあるとそこで終わってしまいます。

ちなみに，ヘルプボックスの表示はモジュール内でプログラムにより実現することもできます。その場合は，後述するCalsリソースを設定することにより，DoHelpメッセージを受け取るように設定します。

●アイコン

コントロールパネルの左下にはアイコンを表示するスペースがあります（図3参照）。ここに，モジュール独自のアイコンを表示することもできます。必要なのは「icl8」，「ICN#」のアイコンリソースと「THUM」リソースです。「THUM」リソースは，表示されるアイコンを設定するとともに，そのアイコンをクリックしたときのアニメーション用に，絵を切り替える速度と，表示するアイコン数，表示するアイコンのリストを設定できるようになっています。THUMリソース用のResEditテンプレートは，ちょっと不具合があるようで，表示するアイコン数が，リストに定義したアイコンの数よりもひとつ少なく設定されてしまうようなのです。一度，HEXEditorで開いて修正してやれば，あとは正常に動いてくれます。

もうひとつ注意しなくてはいけないことは，アイコンのアニメーション，モジュールの切り替えで新しいアイコンを描画する際，前に書かれていたアイコンを消去しないことです。したがって，新しく書かれたアイコンのマスク領域が小さいと，前のアイコンの一部が残ってしまいますので，マスク領域を32×32に設定しておくといいでしょう。

必要メモリ量

AfterDarkがモジュールに割り当てるメモリサイズは，標準で30KBです（古い情報なのでいまはもう少し増えているかもしれません）。もし，作りたいモジュールが，ものすごくメモリを食うものなのであれば，IDが0の



図6　ヘルプボックス（テキスト）

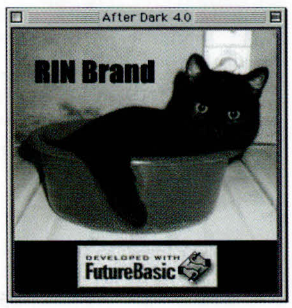


図7　ヘルプボックス（ピクチャ）

「sysz」リソースを作って，メモリ量を設定することにより，割り当てメモリ量を変化させることができます。

どのメッセージを受け取るか

Calsリソースを使うと，モジュールが呼び出されるタイミングを設定することができます。モジュールがAfterDarkから呼び出される際に，message%にその呼び出された理由を表すパラメータが入ってきますが，Calsリソースを使うことにより，それら各メッセージが発生した際にモジュールを呼び出すのか呼び出さないのかを設定することができます。このリソースの実体は，1バイトのフラグセットで，表5のように，各メッセージに対応して，ビットが1になっているメッセージのみを受け取るようになります。ちなみに，このリソースを省略すると，DoInitialize，DoBlank，DoDrawFrame，DoCloseを受け取り，DoSelected，DoHelpを受け取らない設定になります。ちなみに，リソースIDは，1を使います。

サウンド

AfterDarkでは，ver.2以後，モジュール実行中にサウンドを鳴らすこともできるようになっています。そのためにIDが0のChnlリソースを作って，使うサウンドのタイプと，ボリュームを設定します。Chnlリソースのあるモジュールを選択すると，コントロールパネルに，ボリュームを変更するためのボタンが表示されます。SDK付属のドキュメントには，「サウンドタイプとして表6に示した3つのタイプが選択できるが，現在はsampledSynthしかサポートしていない」と書かれています。何度もいうようですが，これはver.3のときの話ですので，現在は不明です。sampledSynthとは「snd 」リソースを演奏することを指しているようです。ただ，演奏ルーチンが，C言語用のライブラリとして提供されていて，FutureBASIC IIからの呼び方がわかりませんでした。SOUND文や，ToolBoxルーチンのSND STARTFILEPLAY関数等を使ってもうまくいきませんので，今後調査を重ねていきたいと思います。

表5　Calsリソースの各ビットの意味

ビット	メッセージ
0	DoInitialize
1	DoBlank
2	DoDrawFrame
3	DoClose
4	DoSelected
5	DoHelp（ResEditのテンプレートでは，DoAboutとなっています）

カラー

clutリソースをモジュールに加えることで，モジュール実行時のカラーテーブルを変更することができます。決まった法則に沿ってIDを設定することにより，モジュールが実行されたときに自動的にそのカラーテーブルを変更してくれます。色深度ごとに複数のclutリソースを設定することができます。色深度が1ビットのときのカラーテーブルは，IDを1001に，2ビットなら1002に，4ビットなら1004に，8ビットなら1008にします。カラーテーブルを設定するとき，白，黒，あとボタンなどで使われているいくつかの色の位置を変更してしまうと，デモモードで実行する際に，コントロールパネルの部分の色が変な色になったりしますので，注意が必要です。

サンプルモジュール

急ぎ足でしたが，AfterDarkモジュール作成に必要な事柄を紹介してきました。ここで，簡単なモジュールのサンプルを示します。このモジュールは，1個から4個のボールが不規則にうごめくという，非常に単純なものですが，ボールの個数，色，移動速度が変更できるようになっています。

全ソースリストを，リスト2に示します。

リソースファイルは「FBBall.res」という名前です。このなかに，ボール

表6　Chnlリソースに設定する値

値	意味
1	noteSynth
3	waveTable
5	sampledSynth

リスト2　サンプルモジュールのソースコード

```
ファイル名：FBBall.bas

'"*****************************************************************************"
'"*   FBBall                                                                  *"
'"*                                                                           *"
'"*    AfterDarkModule For FutureBASICII-J                                    *"
'"*                                          1999/12/12   Takaaki Mizuno      *"
'"*****************************************************************************"

RESOURCES  "FBBall.res",  "ADgmADrk",  "ADgm",  0,  "FBBall",  _resPurgeable
_resSysHeap
'"          ~~~~~"~~~"
'"連結するリソースファイルの名前"
COMPILE 0, _sysHeapVars_pointerVars_dimmedVarsOnly_strResource
OUTPUT FILE "FBBall"
'"             ~~~~~"
'"できあがったモジュールファイルの名前"

'"*************** AfterDarkから渡されるパラメータを定義 ***************"
DIM RECORD monitorData
  DIM monitorRect.8                              '"モニタの矩形"
  DIM synchFlag%                                 '"モニタの垂直同期のフラグ"
  DIM curDepth%                                  '"現在の色深度"
DIM END RECORD.monitorData

DIM RECORD MonitorsInfo
  DIM monitorCount%                              '"接続されたモニタの数"
  DIM monitorList.monitorData                    '"モニタの各データ"
DIM END RECORD.MonitorsInfo

DIM RECORD ctrlArray
  DIM item1%                                     '"第一アイテムの値"
  DIM item2%                                     '"第二アイテムの値"
  DIM item3%                                     '"第三アイテムの値"
  DIM item4%                                     '"第四アイテムの値"
DIM END RECORD.ctrlArray

DIM RECORD qdGlobals
  DIM qdThePort&
  DIM qdWhite.8
  DIM qdBlack.8
  DIM qdGray.8
  DIM qdLtGray.8
  DIM qdDkGray.8
  DIM qdArrow.16
  DIM qdScreenBits&
  DIM qdRandSeed&
DIM END RECORD.qdGlobals

DIM RECORD GMParamBlockRec
  DIM controlValues.ctrlArray                    '"コントロールパネルのアイテムの値"
  DIM monitors&                                  '"モニタデータ用ハンドル"
  DIM colorQDAvail%                              '"カラーQuickDrawは有効か？"
  DIM systemConfig%                              '"システム情報"
  DIM qdGlobalsCopy&                             '"QuickDrawのグローバル変数のコ
                                                 ピーへのポインタ（読み込み専用）"
  DIM brightness%                                '"スクリーンの輝度"
  DIM demoRect.8                                 '"デモモードでのコントロールパネル
                                                 が描画されている矩形"
  DIM errorMessage&                              '"エラーメッセージへのポインタ"
  DIM sndChannel&                                '"サウンド演奏用チャンネル"
  DIM adVersion%                                 '"AfterDarkのバージョン"
DIM END RECORD.GMParamBlockRec

'"AfterDarkからモジュールに渡されるメッセージ(message%)"
'"注：_CloseModuleは，AfterDarkSDKでは_Closeと定義されているが，この定数ではFBの定義済み
```

```
定数とバッティングしてしまう"
_Initialize=0
_CloseModule=1
_Blank=2
_DrawFrame=3

'"モジュールからAfterDarkに渡されるメッセージ"
_ModuleError=-1
_NoProblem=0
_RestartMe=1
_ImDone=2
_RefreshResources=3

DIM gStorageHndl&, gBlankRgn&, gMessage%, gParams&'"AfterDarkから渡されるパラメータ"
DIM gResult%                                      '"AfterDarkに返すパラメータ"

'"ボール移動用変数"
DIM bNumber%
DIM bx%(3), by%(3), bMoveX%(3), bMoveY%(3), bz%(3)

END GLOBALS

'"*************** 関数群 ***************"

'"cicnリソースを描画"
CLEAR LOCAL MODE
DIM rect.8
LOCAL FN gCicn (myLeft,myTop,myRight,myBottom,myPicture)
  DIM myHandle&
  CALL SETRECT (rect,myLeft,myTop,myLeft+myRight,myTop+myBottom)
  myHandle& = FN GETCICON (myPicture)
  LONG IF myHandle&
    CALL PLOTCICON (rect,myHandle&)
    CALL DISPOSCICON (myHandle&)
  END IF
END FN

'"モニタの横幅を取得"
CLEAR LOCAL MODE
DIM tempPramBlock.GMParamBlockRec
DIM tempMonitorBlock.monitorsInfo
LOCAL FN mainMonitorH(paramBlockPtr&)
  BLOCKMOVE paramBlockPtr&,@tempPramBlock,_GMParamBlockRec
  BLOCKMOVE tempPramBlock.monitors&,@tempMonitorBlock,_monitorsInfo
END FN=tempMonitorBlock.monitorList.monitorRect.right%

'"モニタの縦幅を取得"
CLEAR LOCAL MODE
DIM tempPramBlock.GMParamBlockRec
DIM tempMonitorBlock.monitorsInfo
LOCAL FN mainMonitorV(paramBlockPtr&)
  BLOCKMOVE paramBlockPtr&,@tempPramBlock,_GMParamBlockRec
  BLOCKMOVE tempPramBlock.monitors&,@tempMonitorBlock,_monitorsInfo
END FN=tempMonitorBlock.monitorList.monitorRect.bottom%

'"エラーメッセージ"
CLEAR LOCAL MODE
DIM tempPramBlock.GMParamBlockRec
LOCAL FN errorMessage(paramsPtr&,errorText$)
  BLOCKMOVE paramsPtr&,@tempPramBlock,_GMParamBlockRec
  BLOCKMOVE @errorText$, tempPramBlock.errorMessage&, LEN(errorText$)+1
END FN

'"設定されたボールの数をチェック"
CLEAR LOCAL MODE
LOCAL FN number_slider(controlPanelValue%)
  DIM ball_number%
  ball_number% = INT(controlPanelValue%/25)
  IF ball_number%=4 THEN ball_number%=3
END FN=ball_number%

'"初期設定ルーチン"
'"ここで描画作業は行わない！"
CLEAR LOCAL
DIM tempPramBlock.GMParamBlockRec, result_1%
LOCAL FN doInitialize(storage&,blankrgn&,paramsPtr&)
  DIM i%
  result_1%=_noProblem
  '"新しいハンドルに2バイト割り当てますが，このハンドルは使われません。"
  '"これは初期設定中にエラーが起こっていないことを示すためだけに存在しています。"
  gStorageHndl&=FN NEWHANDLE(2)
  '"パラメータブロックを一時記憶領域にコピー"
  BLOCKMOVE paramsPtr&,@tempPramBlock,_GMParamBlockRec
  LONG IF gStorageHndl&<>0
    '"ハンドル取得成功"
    '"ここに初期値の設定を書き込みます。"
    bNumber% = FN number_slider(tempPramBlock.controlValues.item1%)
    FOR i%=0 TO 3
      bx%(i%)=RND(FN mainMonitorH(paramsPtr&) - 35)'"ボールの初期位置(X軸)"
      by%(i%)=RND(FN mainMonitorV(paramsPtr&) - 35)'"ボールの初期位置(Y軸)"
      bMoveX%(i%) = ( RND(3) - 2 )                '"ボールの進む向き(X軸)[-1,0,1]"
      bMoveY%(i%) = ( RND(3) - 2 )                '"ボールの進む向き(Y軸)[-1,0,1]"
      bz%(i%)=RND(20)+20                          '"ボールの進む距離"
    NEXT i%
```

のピクチャが，図8のように，cicnリソースとして，赤，青，緑の順にIDが128～130までに，塗りつぶし用の黒いリソースが131に入っています。すべてサイズは38×38です。また，1つひとつのボールは，図9のように，32×32のボールを黒い領域が包み込んでいます。これは，ボールを動かしたときに，前の位置にあるボールの画像を消す手間を省き，ちらつきを抑えるためのものです。

続いて，コントロールパネルは，図10のようになっています。ボールの数，速度はスライダーで，ボールの色はポップアップメニューで設定を可能にしています。各スライダーのsUntリソースは図11，12のとおりです。ポップアップメニューは上から，「赤」「青」「緑」の順です。

さて，ソースリストですが，それほど特殊なことはやっていないのですが，デモモードで，モジュールを動かしながらスライダーを動かした場合は特にモジュールにメッセージは送られてきませんから，DrawFrameメッセージでのアニメーションだけで，きちんと変化に対応してやる必要があり，そのあたりの処理がポイントかもしれません。

それから，乱数を発生させるRND関数は，FutureBASIC IIのリファレンスマニュアルを見るとコードリソースでは使えないようなことが書いてありますが，なぜか使えます。

DarkSide of the Macintoshについて

この，スターウォーズのパロディっぽい名前のソフトウェアは，シェアウェア版のスクリーンセーバーソフトウェアです（シェアウェア化は5.0から。4.2まではフリーウェアで，こちらも同時に公開されています）。AfterDarkがコントロールパネルなのに比べ，こちらはアプリケーションタイプで，起動項目に入れて使います。アプリケーションタイプゆえに，こちらを好んで使う人も多いようです。DarkSide of the Macintoshは，AfterDarkと同じように，Faderと呼ばれるファイルを追加することで，さまざまなスクリーンセーバーが楽しめます。Faderも，やはりAfterDarkのモジュールと同様にコードリソースで，そのほかコントロール用に持っているいくつかの

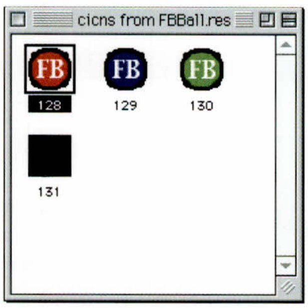


図8　サンプルのcicnリソース

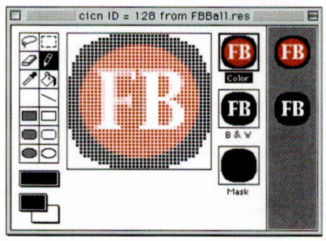


図9　ボールの画像



図10　サンプルのコントロールパネル

```
  XELSE
    'ハンドル取得失敗""
    result_1%=_moduleError
    FN errorMessage(paramsPtr&,"メモリが足りません...")
  END IF
END FN=result_1%

'"はじめての描画"
CLEAR LOCAL
DIM tempPramBlock.GMParamBlockRec
LOCAL FN doBlank(blankrgn&,paramsPtr&)
  DIM i%
  BLOCKMOVE paramsPtr&,@tempPramBlock,_GMParamBlockRec
  CALL PAINTRGN(blankrgn&)
  FOR i%=0 TO 3
    LONG IF bNumber% >= i%
      FN gCicn (bx%(i%),by%(i%),38,38,127+tempPramBlock.controlValues.item3%)
    END IF
  NEXT i%
END FN=_noProblem

'"モジュール終了"
CLEAR LOCAL
LOCAL FN doClose(storage&,blankrgn&,paramsPtr&)
END FN=FN DISPOSHANDLE(gStorageHndl&)

'"アニメーション"
CLEAR LOCAL
DIM tempPramBlock.GMParamBlockRec
LOCAL FN drawFrame(blankRgn&,paramsPtr&)
  DIM result_1%, tempX%, tempY%, oldX%, oldY%, i%
  DIM tempNumber%
  result_1%=_noProblem
  '"パラメータブロックを一時記憶領域にコピー"
  BLOCKMOVE paramsPtr&,@tempPramBlock,_GMParamBlockRec

  '"現在のボール数の設定をチェック"
  '"少なくなってる場合は，余分なボールを消す"
  tempNumber% = FN number_slider(tempPramBlock.controlValues.item1%)
  LONG IF tempNumber% < bNumber%
    FOR i%=0 TO 3
      LONG IF tempNumber% < i%
        FN gCicn (bx%(i%),by%(i%),38,38,131)
      END IF
    NEXT i%
  END IF
  bNumber% = tempNumber%

  '"ボールを移動"
  FOR i%=0 TO bNumber%
    LONG IF bz%(i%) > 0
      oldX% = bx%(i%)
      oldY% = by%(i%)
      tempX% = bx%(i%) + ( bMoveX%(i%) * ((tempPramBlock.controlValues.item2% /
34 ) + 1))
      tempY% = by%(i%) + ( bMoveY%(i%) * ((tempPramBlock.controlValues.item2% /
34 ) + 1))
      LONG IF tempX% < 0 OR (tempX% > (FN mainMonitorH(paramsPtr&)) - 38)
        bMoveX%(i%) = bMoveX%(i%) *(-1)
        tempX% = bx%(i%)
      END IF
      LONG IF tempY% < 0 OR (tempY% > (FN mainMonitorV(paramsPtr&)) - 38)
        bMoveY%(i%) = bMoveY%(i%) *(-1)
        tempY% = by%(i%)
      END IF
      FN gCicn (tempX%,tempY%,38,38,127+tempPramBlock.controlValues.item3%)
      bx%(i%) = tempX%
      by%(i%) = tempY%
      DEC(bz%(i%))
    XELSE
      FN gCicn (bx%(i%),by%(i%),38,38,127+tempPramBlock.controlValues.item3%)
      bMoveX%(i%) = ( RND(3) - 2 )
      bMoveY%(i%) = ( RND(3) - 2 )
      bz%(i%)=RND(20)+20
    END IF
  NEXT i%
END FN=result_1%


'"*************** メインルーチン ***************"
ENTERPROC% (gStorageHndl&, gBlankRgn&, gMessage%, gParams&)
  SELECT gMessage%
    CASE _Initialize
      gResult%=FN doInitialize(gStorageHndl&, gBlankRgn&, gParams&)
    CASE _CloseModule
      gResult%=FN doClose(gStorageHndl&, gBlankRgn&, gParams&)
    CASE _Blank
      gResult%=FN doBlank(gBlankRgn&, gParams&)
    CASE _DrawFrame
      gResult%=FN drawFrame(gBlankRgn&, gParams&)
  END SELECT
EXITPROC%=gResult%
RETURN
```

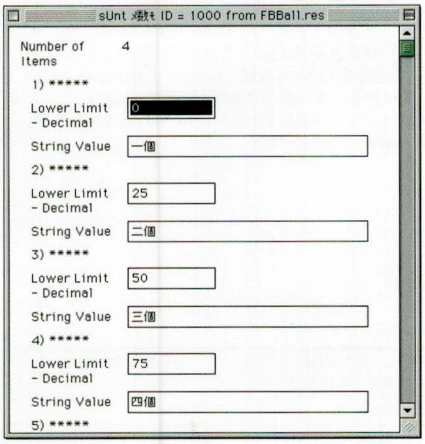


図11 「数」スライダーのsUnt リソースパネル

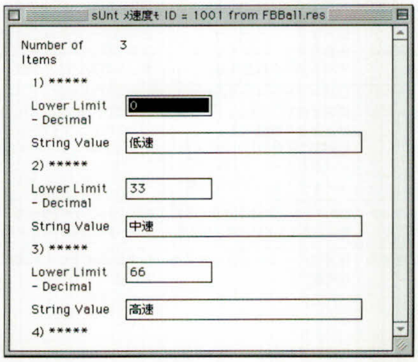


図12 「速度」スライダーの sUntリソースパネル

リソースも，名前，構造こそAfterDarkのものとは異なるものの，基本的な考え方はまったく同じです。DarkSide of the Macintoshにも，Faderを作るためのサンプルコードもついています。

しかも，DarkSide of the Macintoshは，なんとAfterDarkのモジュールも使えてしまうのです。AfterDarkのモジュールをFaderといっしょに，アプリケーションのフォルダに入れておけば，Fader一覧にちゃんと登場してくれ，選択すれば設定の変更もきちんとできるようになっていました。

互換性が，どこまで完全なものかはわかりませんが，少なくとも私がFutureBASIC IIで作成したモジュールはきちんと動いてくれました。モジュールは作ってみたいけど，手元にAfterDarkがない！　とお嘆きの方は，一度試してみてはいかがでしょうか。

DarkSide of the Macintoshは以下のURLでダウンロードできます。

http://www.poubelle.com/

おわりに

ここまで，AfterDarkモジュールの作り方について，ごく簡単に説明してきましたが，いかがだったでしょうか。アイデア次第でさまざまなモジュールが作れますし，AfterDarkは今後も進歩を続けていくだろうと思いますので，いろいろなものにチャレンジしてみてください。

ところで，AfterDarkのパッケージにも，モジュールはたくさん付属していますし，フリーのモジュールも公開されています。もしかしたら，モジュールを開発する必要は実はあまりないのかもしれません。しかし，どんなものでも作れるものがあれば作ってみたくなるのがプログラマの性。この解説が，そんなみなさまの魂に火をつけることに少しでも役立ち，今後もさまざまなモジュールが公開されれば幸いです。

May The Code Be With You.

特別企画

西暦2000年のMIDI環境

GM2のひみつ

西川善司　Nishikawa Zenji

1999年に登場したSC-8850をはじめ，今後高いレベルで音源を標準化していくと期待される「General MIDI System Level 2」すなわちGM2規格。ここではその仕様を確認するとともに，新しい音源の方向性を追ってみたい。

■GM1→GM2への流れ

1991年10月にJMSC（日本MIDI規格協議会）とMMA（米国のMIDI規格管理組織）によって制定された「GMシステムレベル1」（以下GM1）は，MIDI音源の標準仕様として迎えられたほか，PC向け音源カードのウェーブテーブル音源機能の標準仕様としても取り入れられ，瞬く間に浸透した。

ところが，その表現力の限界がすぐに見え始めたためか，GM1対応製品は発売されるものの，ユーザーが積極的に活用する状況はあまり多く見られなかった。

たとえばPCの世界で見てみると，これだけGM1対応の音源カードが浸透しながらも，GM1対応のアプリケーションというとPC通信カラオケくらいのものだ。もっとも利用されると見込まれていたPCゲームのBGM演奏用音源としては対応状況はいまひとつで，結局，CD-DAあるいはその他の方式による「オーディオストリームの垂れ流し」というスタイルのほうがメインになってしまった。

この状況を改善していこうと立ち上がったのが，大手楽器メーカーであるローランドとヤマハの2社だ。ローランドは「GSフォーマット」，ヤマハは「XGフォーマット」という名称の，GM1を独自拡張した音源仕様を自社製品に適用し，どんどん新製品を投入していくことになる。[*1]

現在ではMIDI音源の標準仕様というと，実質的にはローランドGSあるいはヤマハXGという大手メーカーの仕様を指すことになっており，標準規格化団体がせっかく定めた「GM1」は上辺だけの仕様となってしまった。

注（*1）実際にはGSはGM1よりも早く発表されている。初代GS音源SC55の初期ロットにはGMマークがついていない。

■GM2登場

この状況を改善するために，1999年5月に発表されたのが「General MIDI Level2 Recommended Practice[*2]」（以下GM2）だ。

GM1とGM2，およびXG，GSを比較した表を表1に示す。

音色数や同時発音数が増えているのは時代の流れを考慮すれば，まあ当然の流れとしても，リズムチャンネルの複数化への対応，エフェクト機能コントロールのサポートや，コントロールチェンジ，エクスクルーシブメッセージがサポートされた点は注目に値する。

これまで，GM1は音源（ハードウェア）仕様最低基準というイメージだったのが，GM2では曲データ制作者側の立場に立った，「制作サイドが不満に思わない程度の表現力を持った音源仕様はこれくらいあれば大丈夫です」という感じの，現在の主流のDTM音源（とどのつまりはGS，XGを指す）の平均スペックを示したようなものになっている。

全体として見比べてみると，スペック的にはだいたいGS音源の最低レベルラインに近いものということができそうだ。

注（*2）「Recommended Practice（奨励事項）」という表現からわかるとおりGM2もGM1と同じく厳密には「規格」ではない。「GM規格」という表現は厳密には間違いということになる。しかし，「GM」のロゴを獲得するためにはこの奨励事項を満足していなければならないので，結局ロゴを獲得する側にとっては規格のようなものである。

表1　各音源と規格の比較

	GM1	GM2	GS（ローランド）	XG（ヤマハ）	ローランド最新GS音源「SC-8850」	ヤマハ最新XG音源（MU128）
音色数	128以上	256以上	187以上	480以上	1640	1342
ドラムセット	1以上	9以上	9以上	11以上	63	47
同時発音数	24以上	32以上	24以上	32以上	128	128
リズムチャンネル	チャンネル10のみ	チャンネル10と11を指定可能	基本はチャンネル10だが，任意のチャンネルをリズムチャンネルに指定可能	基本はチャンネル10だが，任意のチャンネルをリズムチャンネルに指定可能	基本はチャンネル10だが，任意のチャンネルをリズムチャンネルに指定可能	基本はチャンネル10だが，任意のチャンネルをリズムチャンネルに指定可能
エフェクタ	なし	リバーブとコーラス	基本はリバーブとコーラスだが，機種によってさまざまなものを搭載	リバーブとコーラスのほか，最低で合計3系統以上のエフェクトを装備する。うち1系統は特定のパートのみにかけられるインサーションエフェクトとして設定可能	リバーブ（8種類），コーラス（8種類），ディレイ（10種類），2バンドイコライザー，インサーションエフェクト（64種類）	リバーブ（12種類），コーラス（14種類），バリエーション（70種類），5バンドイコライザー（5種類），インサーションエフェクト1（43種類），インサーションエフェクト2（43種類）
コントロールチェンジ	ごく基本的な演奏表現に関係するパラメータのみ設定変更可能	音色パラメータの設定変更や演奏表現の変化などの設定が行える	音色パラメータの設定変更や演奏表現の変化などの設定が行える	音色パラメータの設定変更や演奏表現の変化などの設定が行える	音色パラメータの設定変更や演奏表現の変化などの設定が行える	音色パラメータの設定変更や演奏表現の変化などの設定が行える
エクスクルーシブ	初期化のメッセージのみ規定されている	ユニバーサルシステムエクスクルーシブメッセージがサポートされ，マスターボリュームやエフェクタのパラメータが制御できる	音源側のメモリを随時書き換え可能	音源側のメモリを随時書き換え可能	音源側のメモリを随時書き換え可能	音源側のメモリを随時書き換え可能

■GM2の特徴

GM2はGM1からどう拡張されたのか，具体的に見ていくことにしよう。

★音色とドラムセット

音色の管理方法

GM2の音色はGM1の基本128種類からさらに128種類を加えた2倍の256種類に増加した。

ところで，MIDIメッセージは8ビットコード体系で，データは7ビット表現であるため，128以上の大きい値を表現できない。「では，どうやって256個もの音色を管理するのか」という疑問がわく。

これは1983年のMIDI規格制定からかなりあとになって追加された「バンク切り換え」のコントロールチェンジメッセージ(*3)を駆使して行っている。

つまり，GM1では音色を，

音色番号:1-128 (0-127)
※ () 内は内部表現 (以下同様)

だけで指定していたのが，GM2では，

バンク番号:0-127
音色番号:1-128 (0-127)

で表現することになる。

実際にはバンク切り換えによって管理されるバンク数は128×128 (14ビット) であるため，正確には音色は，

バンク番号:0-32767
音色番号:1-128 (0-127)

で表現される。しかしその14ビットで表現されるバンク番号の上位7ビットは，GM2では「属性」的な扱われ方をするので (この「属性」については後述)，実質的には音色は，

バンク番号:0-127
音色番号:1-128 (0-127)

で管理されていると考えていいだろう。ちなみにバンク0に含まれる音色番号1-128 (0-127) はGM1で規定されている128個の音色に相当する。新しく追加された音色はバンク1以降の音色番号1-128 (0-127) にあることになる。

この表現方法だと128×128で32768個の音色が取り扱えることになるので，「では，GM2になって追加された128個の音色はどのバンクに格納されているの？」という興味もわくはずだ。

しかし，実は128個がまるまるどこかのバンクに入っているのではなく，「ある音色番号の楽器音の別バリエーションを別バンクに格納する」というスタンスで追加されているのだ。

たとえば，バンク0:音色番号5 (4) は「Electric Piano1」だが，これの別バリエーションとして，

バンク1:音色番号5 (4)
Detuned Electric Pinao1
バンク2:音色番号5 (4)
Electric Piano1 (velocity mix)
バンク3:音色番号5 (4)
60's Electric Piano

のように，別バンクに同じ種類の楽器の別タイプ (あるいは別奏法，別の表現方法) の音が収録されているのだ。

GM2の音色リストをリスト1に示す。

こうした配列的な音色管理方式により，256個のなかからひとつの音色を探す必要はなく，128個の音色のなかからまずひとつの「音色ジャンル」を選び，そこから別バンクをあたり，好みの音色を探す……といった音色の探し方になるわけだ。

なお，音色番号14 (13) 「Xylophone」のように，音色番号によってはGM2になってもまったく別のバリエーションが追加されていない音色もある。

注 (*3) 音源制御用のMIDIメッセージのひとつ。

音色の切り換え方

バンク切り換えによる音色管理方法はGS音源，XG音源ですでに採用されているので，実際の切り換え方はそうした音源のユーザーならば戸惑わずにできそうな気がする。実際，切り換えの際に使用するMIDIメッセージはGM2もGSもXGもまったく同じだ。ところが，GM2のバンク番号の使い方は，微妙にGS，XGとは異なっているのである。

バンク切り換えのMIDIメッセージはコントロールチェンジ0番と32番を使う。0番はバンク番号の上位バイト，32番は下位バイトの送信に使う。

GM2ではバンク番号の上位バイトは「属性バイト」のように使われるので実際のバンク番号は下位バイトを使って送信することになる。

GS音源ユーザーは気づいたと思うが，この時点でGSとはバンクの切り換え方が違うのだ。

GSではバンクの切り換えは下位バイトを一切使わず，上位バイトのみで行う仕様であるため，GSとGM2の音色管理方式は似て非なるものなのだ。

さて，この「属性バイト」だが，GM2で現在規定されているのは120と121の2つ。120が「そのチャンネル (パート) をリズム (ドラム) にする」という意味を持っているのである。これに対し121は「そのチャンネルを，通常の楽器音チャンネルにする」という意味を持つ。

いまのところ，GM2では，このバンク切り換えの上位バイト (「属性バイト」) を認識するのは「MIDIチャンネル11だけ」としているのでほかのパートではなにを送っても影響はないはずだ。が，今後登場するであろうGM3以上でどう規定されるかわからないので，音色切り換えの際には，現在意味がなくても上位バイトと下位バイトを送るべきだろう。

もっとも同一バンク内の音色切り換えであればバンク切り換えコマンドの再送信は不要なので，曲の最初に「属性バイト」を送信してしまえば，以降の音色切り換えの際にはバンク番号の下位バイトのみの送信だけでバンク指定ができることになる。

それでは，実際にどういうMIDIメッセージを送ったらいいかを解説していこう。

たとえばMIDIチャンネル7にバンク番号2の音色番号15 (14) の切り換えを指示する場合は，

B6　チャンネル7へコントロールチェンジ指定
00　バンク切り換え上位バイト
79　属性バイト＝121 (通常楽器チャンネル)
B6　チャンネル7へコントロールチェンジ指定
20　バンク切り換え下位バイト
02　バンク番号
C6　チャンネル7へ音色切り換え指定
0E　音色番号15 (14)

ということになる。

リズムセットの切り換え方

GM1では，MIDIチャンネル10を半ば強制的にリズムチャンネルとして割り当てられ，たったひとつのリズムセットしか持っていなかった。これがGM2ではリズムセットが8セットも追加され，合計9リズムセットが利用可能になり，さらに前述の「属性バイト」の設定で，MIDIチャンネル10を通常楽器パートにしたり，MIDIチャンネル11をリズムチャンネルとして利用できるようになっている。

GM2で新しくなったリズムセットのリストをリスト2に示す。

さて，まず，リズムセットの切り換え方だが，これはリズムチャンネル上で音色切り換えを行うだけでよい。GSやXGと全く同一のやり方だ。たとえば，リズムセットをELECTRICセットに切り換えたい場合は音色切り換えで音色番号25を指定すればいいことになる。

ところで，すでに述べたようにGM2では，MIDIチャンネル11に対し，バンク切り換えの上位バイトを120にして送信すると，このチャンネルをリ

ズムチャンネルにすることができるようになった。この設定を行うと，以後，MIDIチャンネル11でも音色切り換えを行えば任意のリズムセットを利用することができるようになる。これまではGS，XGでしかできなかった多彩なリズム表現がGMでもできるようになったのだ。

ここで，ひとつの疑問が浮上する人もいるだろう。属性バイトを120として送信した場合，バンク切り換えの下位バイトにはなにを送ったらいいのかという疑問だ。

実は現在，これはGM2では規定されていないのである。つまりリズムチャンネルでは，バンク切り換えの下位バイトは無視されてしまうのだ。ただ，「デフォルトでは0を設定するように」とGM2の仕様書には書かれているため，今後を考えるならば，下位バイトとして0も送信しておくべきだろう。

それでは具体的な例を示す。

MIDIチャンネル10のリズムセットを音色番号25（24）「ELECTRIC」に切り換える場合は，

B9　チャンネル10へコントロールチェンジ指定
00　バンク切り換え上位バイト
78　属性バイト＝120（リズムチャンネル）
B9　チャンネル10へコントロールチェンジ指定
20　バンク切り換え下位バイト
00　バンク番号（ダミー）
C9　チャンネル10へ音色切り換え指定
18　音色番号25（24）

という感じになる。MIDIチャンネル10は楽器を初期化した段階でバンク設定は行われているはずなので，実際のシーケンス時にはバンク設定は不要だろう。

なお，属性バイトを121に設定すれば，MIDIチャンネル10を通常の楽器チャンネルに設定することも可能だ。

もうひとつ，MIDIチャンネル11をリズムチャンネルに設定し，ここにリズムセットとして音色番号57（56）「SFX」を呼び出す場合の例を示そう。

BA　チャンネル10へコントロールチェンジ指定
00　バンク切り換え上位バイト
78　属性バイト＝120（リズムチャンネル）
BA　チャンネル10へコントロールチェンジ指定
20　バンク切り換え下位バイト
00　バンク番号（ダミー）
CA　チャンネル10へ音色切り換え指定
38　音色番号25（24）

なお，データ制作者として注意したいのは，MIDIチャンネル11をリズムチャンネルとした場合「もたつきがあるかもしれない」という点だ。

GM音源ではリズムの演奏がもたつくのを懸念してMIDIチャンネル10を最優先に処理しており，たとえばGS音源やXG音源の場合，

高←　チャンネル処理優先順位　→低
10 1 2 3 4 5 6 7 8 9 11 12 13 14 15 16

という優先順位で各チャンネルを処理している。そう，MIDIチャンネル11のチャンネル処理優先順位はMIDIチャンネル10に比べてかなり低いのだ。よってリズムセットを複数使った曲を作る場合には主リズムをMIDIチャンネル10で，サブとなるリズムをMIDIチャンネル11で処理したほうがいい。

GM3ではGSやXGのように任意のチャンネルをリズムに設定できるようにしてほしいものだ。

★エフェクタ

エフェクタの制御ができるようになったGM2

GM1ではエフェクタの制御に関するコントロールチェンジなどが一切規定されていなかったのだが，たいていのGM1音源はリバーブとコーラス程度のエフェクト機能を内蔵していた。

GM2はこの実情に従い，音源に内蔵されるエフェクタの制御に関するコード体系が新設された。

コントロールチェンジによるエフェクタの制御

これはGSやXGで使われている，エフェクタ制御用のコントロール・チェンジの定義をそのままGM2が継承する形となった。具体的には，

91　リバーブセンドレベル
93　コーラスセンドレベル

というふうに規定された。

94はGSではディレイセンドレベルにアサインされているがGM2ではこれは取り込まれていない。またちょうど挟まれている92番も空きのままだ(*4)。

具体的にはこのパラメータは各パートのエフェクトのかかり具合を設定するものだ。面白いのはこれらのデフォルト値で，リバーブが40，コーラスが0……という，まったくGS音源と同じデフォルト値になっているのだ。

注(*4) MIDI規格1.0によれば92番には「トレモロデプス」にリザーブされているように思えるが……。

エクスクルーシブによるエフェクトの制御

コントロールチェンジによるエフェクトパラメータ制御は，せいぜい各チャンネルのエフェクタのかかり具合を設定できる程度のものだ。通常はこれでも十分だが，より高度な曲データを作ろうとすると，エフェクトセットの切り換えに手を出したくなるものだ。

この要望に応えるべく，GM2ではついにエフェクトセットが規定されるに至った。

制御方式には「ユニバーサルエクスクルーシブメッセージ」と呼ばれる楽器メーカーや機種不問の共通エクスクルーシブメッセージを利用する。

切り換えられるエフェクトセットのタイプは，やはり，リバーブとコーラスのみだ。

具体的にその内容を示そう

F0 7F Universal Real-Time SysEx header
<DevID> デバイスID（7F ＝ all devices）
04 sub-ID#1 ＝ DEVICE CONTROL
05 sub-ID#2 ＝ GLOBAL PARAMETER CONTROL
01 Slot Path Length ＝ 1
01 Parameter ID Width ＝ 1
01 Value Width ＝ 1
01 Slot Path MSB ＝ 1
01 Slot Path LSB ＝ 1（Effect 0101: Reverb）
pp Parameter to be controlled.
vv Value for the parameter.
F7 EOX

Slot Pathというのが，ローランドやヤマハのGS，XG音源でいうところの「パラメータアドレス」に相当するものと考えていい。

「Slot Path＝0101」でリバーブセットの切り換え指定，「Slot Path＝0102」でコーラスセットの切り換え指定となる。

そしてGM2で規定されたエフェクタは以下のとおりだ。

・リバーブ
0: Small Room
1: Medium Room
2: Large Room
3: Medium Hall
4: Large Hall
8: Plate
・コーラス

0: Chorus 1
1: Chorus 2
2: Chorus 3
3: Chorus 4
4: FB Chorus
5: Flanger

リバーブは5がなくて8にPlateがアサインされている点に注意。5～7は別のエフェクタタイプが今後追加されることを予感させる。

実際の設定変更時にはppには上記のエフェクタのタイプ番号を，vvにエフェクトパラメータを設定する。

エフェクトパラメータvvはリバーブの場合は残響時間のパラメータになる。0がもっとも短く，127がもっとも長い。

コーラスの場合はコーラスタイプによってパラメータの効能は変わってくるため，詳しくはGM2の仕様書（http://www.amei.or.jp/report/AMEI_MMA/gm2v03d.pdf）や各GM2音源の仕様を見てほしいが，値が大きいほど，過激にコーラス効果が効いてくると考えていい。

なお，デフォルトのエフェクトセットはリバーブがタイプ4（Large Hall），コーラスがタイプ2（Chorus 3）となっている点に注意してほしい。

具体的なメッセージ送信例も示しておこう。

デバイスID＝0，切り換えるエフェクトをpp＝8（plate），残響時間パラメータを50（＝$32）に設定する場合は，

```
F0 7F 00 04 05 01 01 01 01 01 08 32 F7
                              ↑  ↑
                              pp vv
```

という感じになる。

★コントロールチェンジ

大幅に新規定が追加されたコントロールチェンジ群

エフェクタ制御用だけでなく，GM2ではかなりの数のコントロールチェンジが追加されている。GM1では，

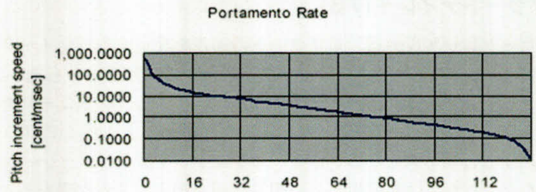
1	モジュレーション
7	ボリューム
10	パンポット
11	エクスプレッション
64	ホールド（ダンパー）
6/38	データエントリ
100/101	RPNのLSB/MSB

だけだったが，GM2では新たに，

0/32	バンク切り換えのMSB/LSB
5	ポルタメントタイム
65	ポルタメント　オン/オフ
66	ソステヌート
67	ソフト
71	フィルタレゾナンス（ハーモニックコンテント）
72	リリースタイム
73	アタックタイム
74	ブライトネス（カットオフ）
75	ディケイタイム
76	ビブラートレート
77	ビブラートデプス
78	ビブラートディレイ
91	リバーブセンドレベル
93	コーラスセンドレベル

が追加された。

順番に見ていくことにしよう。



●ポルタメントタイム(5)，ポルタメント(65)

ポルタメント（65）をオンにされたチャンネルは，ポルタメントタイム（5）で指定されるピッチ上昇率に従いピッチを上昇させる。0がもっとも高速（高上昇率）で，127が低速（低上昇率）となる。

ポルタメント（65）のオン，オフのパラメータ値は，

0-63	オフ
64-127	オン

に対応するが，MIDIの世界において，こうしたスイッチング系パラメータの設定は慣例的に，

0＝オフ
127＝オン

として行うのが普通だ。

この仕様はGSやXGと同じなのだが，ポルタメントのソースノートの指定方法が規定されていないのが少々謎ではある。GSやXGではコントロールチェンジ84番で指定するのだが，GM2にはこれに関する記載がない。単なる記載漏れだろうか。

●ソステヌート(66)

オンにすると，その時点で鳴っている音のみを持続させる効果をもたらす。ダンパー（ホールド）の効果に近い。

●ソフト(67)

オンにしている間に発音した音は，すべてぎらつき感のなくなった柔らかな音になる。

効果自体の実現方式は音源，あるいは音色のタイプによって自由に処理してよいことになっているが，カットオフ周波数を低くして表現する手法がよく用いられるようだ。

●フィルタレゾナンス(ハーモニックコンテント)(71)，ブライトネス(カットオフ)(74)

シンセサイザ側のレゾナンスパラメータやカットオフ周波数を制御するコントロールチェンジ。どうやらコントロールチェンジ71～74はXGからGM2にインポートされたもののようだ。

レゾナンス（71）のパラメータは64を中央値として0がもっともレゾナンス効果を弱め，127がもっともレゾナンス効果を強める設定となる。具体的な処理内容についてはこれも「ソフト（67）」同様に，各音源やそのコンセプトに任せる……ということになっている。

ブライトネス（カットオフ）（74）の場合は64を中央値として，0が周波数をもっとも低くする設定に対応し，127が周波数をもっとも高くする設定に対応する。これも具体的な効果は各音源が処理していいことになっているが，一般にカットオフ周波数を広くすると音は柔らかくなり，高くするとぎらつき感が出てくる傾向にある。

●リリースタイム(72)，アタックタイム(73)，ディケイタイム(75)

エンベロープパラメータの設定をするコントロールチェンジ。（図）

64を中央値とし，0がもっとも短く，127がもっとも長い設定になる。

アタックタイムは発音後，最高レベルになるまでの時間を表し，ディケイタイムは最高レベルからの減衰時間のことを指す。リリースタイムはノートオフ受信後の余韻をどのくらい残すかを指定するものだ。

●ビブラートレート(76)，ビブラートデプス(77)，ビブラートディレイ(78)

ビブラートレート(76)は，ビブラートの速さを設定するパラメータだ。64を基準値とし0がもっとも揺らぎの速度が遅く，127がもっとも速い。

ビブラートデプス(77)はビブラートの深さ設定するコントロールチェンジだ。64を基準値とし，0がもっとも浅く，127がもっとも深くビブラートの効果を出すことができる。コントロールチェンジ1番のモジュレーションに似ているが，こちらは基準値からの相対表現で指定する点が異なる。

ビブラートディレイ(78)は実際にビブラートがかかるまでの時間の遅れを設定するものだ。これも相対表現で指定することになっており，64が基準値，0がもっともディレイが短く，127がもっとも長い。「なにをトリガにしてディレイをカウントするのか」が明確にされていないが，これは各GM2音源メーカーが自由に設定していいことになっているようだ。おそらくノートオンメッセージの受信をトリガにするパターンが一般的に用いられると予想される。

さて，疑問として上るのは「コントロールチェンジ1番のモジュレーションとなにが違うのか」ということではないだろうか。実はこれについてはGM2では明確に規定されておらず，「ビブラート」で制御される効果についても楽器メーカーが自由に行っていいことにしている。

混乱を避けるためにも両コントロールチェンジの関係を明確にする規定を早く作ってほしいものだ。そうでないと，同じ曲データでも，演奏に使用するGM2音源によってビブラート(モジュレーション)がかかったりかからなかったりする事態が起こりうる。

●リバーブセンドレベル(91)，コーラスセンドレベル(93)「エフェクタ」の段を参照してほしい。

★エクスクルーシブ

本来，エクスクルーシブメッセージとは各楽器メーカーが自由に仕様を策定してよいものとして定義されたものだが，のちに一般的なパラメータの制御手法は「ユニバーサルシステムエクスクルーシブメッセージ」(以下USXM)としてMIDI規格のなかにも取り込まれることとなった。

GM1におけるUSXMはGM1音源の初期化メッセージのみの規定だったがGM2ではパラメータの設定変更などの手段も規定された。

ユニバーサルリアルタイムシステムエクスクルーシブメッセージ

●マスターボリューム，マスターファインチューニング，マスターコースチューニング

マスターボリュームはその名のとおり，音源の最終的な音量調整を行うパラメータだ。

そしてマスターファインチューニング，マスターコースチューニングは，その音源のピッチ(音高)のチューニングを行うためのパラメータである。

これらの設定には，

```
F0 7F Universal Real-Time SysEx header
<DevID> デバイスID (7F = all devices)
04 sub-ID#1 = DEVICE CONTROL
xx sub-ID#2 = PARAMETER TYPE
ll Parameter to be controlled.
mm Value for the parameter.
F7 EOX
```

というメッセージフォーマットを用いる。

xxでパラメータタイプの指定を行い，ll，mmにそれぞれそのパラメータに対する設定値の下位バイト，上位バイトを与えることになる。

・xx＝01 マスターボリューム

xx＝01のときマスターボリュームの設定になり，値の設定範囲はll，mm＝00，00～$7f，$7fとなる。初期値にはll，mm＝$7f，$7fの最大値が設定される。音源によっては下位バイトを無視して上位バイトのみ認識するものもある。

・xx＝03 マスターファインチューニング

xx＝03のとき，マスターファインチューニングの設定になる。マスターファインチューニングとはその音源の発音ピッチの微調整のことだ。

設定値ll，mm
ll，mm＝$00，$00～$00，$40～$7f，$7f

は，

-100セント～中央(ニュートラル)～+99.9セント

に対応する。なお，1セントは半音の100分の1に相当する(すなわち100セント＝1半音)。

・xx＝04 マスターコースチューニング

xx＝04のとき，マスターコースチューニングの設定になる。マスターコースチューニングとはその音源の発音ピッチの調整を半音単位で行うことをいう。

設定値ll，mmのうちllは無視され，

mm＝$28～$40～$58

を設定有効範囲とし，これが

-24半音～中央
(ニュートラル)～+24半音

に対応する。

●リバーブパラメータ，コーラスパラメータ

「エフェクタ」の項を参照してほしい。

●チャンネルプレッシャー(アフタータッチ)のセッティング

GM2で追加されたUSXMの中でもっともユニークなのが，この「チャンネルプレッシャー(アフタータッチ)のセッティング」だ。

これによりアフタータッチが行われたときにどのような効果をもたらすかを自在にユーザーが設定できるのだ。

通常，MIDIで情緒表現を行おうとするには，さまざまな演奏表現に対応するコントロールチェンジを1つひとつ制御する必要があった。たとえば，「ビブラートをかけながらポルタメントする」といった表現を行いたい場合は，ビブラート(コントロールチェンジ1番)とポルタメント(コントロールチェンジ65番)を送信する必要がある。

ところが，このセッティングを行うとアフタータッチを送信するだけで，ビブラートやポルタメントの制御を同時に行えてしまうのだ。

メッセージフォーマットを以下に示そう。

```
F0 7F Universal Real Time SysEx header
<DevID> デバイスID (7F = all devices)
09 sub-ID#1 = Controller Destination Setting
01 sub-ID#2 = Controller Type: 01 (Channel Pressure)
0n MIDI Channel (00 - 0F)
[pp rr] Controlled parameter and range
  :
F7 EOX
```

nには操作対象のMIDIチャンネルを0～Fの範囲で入れてやる。

ppとrrは，それぞれ，コントロールパラメータ番号とその設定値だ。表

表2

コントロールパラメータ(pp)	設定範囲(rr)	機能範囲	デフォルト値
00 Pitch Control	$28～$40～$58	-24半音～中央～+24半音	$40
01 Filter Cutoff Control	$00～$7F	9600～+9450セント	$40
02 Amplitude Control	$00～$7F	0～(127/64)×100%	$40
03 LFO Pitch Depth	$00～$7F	0～600セント	$00
04 LFO Filter Depth	$00～$7F	0～2400セント	$00
05 LFO Amplitude Depth	$00～$7F	0～100%	$00

2参照。

ppは現在00～05の6種類があるが，実際の設定時には必ずしも6個すべてについて設定する必要はない。

具体例を示しておこう。たとえば，いま，全GM2音源に対し（デバイスID＝$7F），チャンネル3（内部表現＝2）でアフタータッチを行ったときに，ピッチモジュレーション（03 LFO Pitch Depth），アンプリチュード・モジュレーション（05 LFO Amplitude Depth）の効果を出すように，設定したい場合には，

F0 7F Universal Real Time SysEx header
7F デバイスID（7F ＝ all devices）
09 sub-ID#1 ＝ Controller Destination Setting
01 sub-ID#2 ＝ Controller Type: 01（Channel Pressure）
02 MIDI Channel3
03 LFO Pitch Depth
40 300セント
05 LFO Amplitude Depth
40 50%
F7 EOX

のように設定すればいい。

コントロールチェンジのセッティング

前出の「アフタータッチのセッティング」に非常に似ているもので，任意のコントロールチェンジ番号に，特殊演奏表現のコントローラを割り当ててしまおう，というのが，この「コントロールチェンジのセッティング」だ。

F0 7F Universal Real Time SysEx header
<DevID> デバイスID（7F ＝ all devices）
09 sub-ID#1 ＝ Controller Destination Setting
03 sub-ID#2 ＝ Controller Type: 03（Control Change）
0n MIDI Channel（00 - 0F）
cc Controller Number（01-1F，40-5F）
[pp rr] Controlled parameter and range
:
F7 EOX

0nは操作対象MIDIチャンネル，ccが割当先コントロールチェンジ番号，pp rrは前出と同じで，コントロールパラメータ番号とその設定範囲だ。

割当先コントロールチェンジは$00-$1F，$40～$5Fであればどれでもよく，GM2ですでに機能定義されているコントロールチェンジを選択することも許されている。

さて，今，たとえば，全GM2音源に対し（デバイスID＝$7F），チャンネル5（内部表現＝04）においてコントロールチェンジ3番（cc＝03）に，ピッチモジュレーション（03 LFO Pitch Depth），アンプリチュード・モジュレーション（05 LFO Amplitude Depth）の制御機能を割り当てたいという場合，送信MIDIメッセージは，

F0 7F Universal Real Time SysEx header
7Fデバイス ID（7F ＝ all devices）
09 sub-ID#1 ＝ Controller Destination Setting
03 sub-ID#2 ＝ Controller Type: 03（Control Change）
04 MIDI Channel5
03 Controller Number＝03
03 LFO Pitch Depth
40 300セント
05 LFO Amplitude Depth
40 50%
F7 EOX

となる。

なお，pp rrのペアはひとつでもいいし，パラメータ番号00～05の6個すべてをひとつのコントロールチェンジに割り当ててしまうこともできる。

●スケール/オクターブチューニング

音律の設定が行えるのがこのUSXMだ。具体的には，

C C# D D# E F F# G G# A A# B

の12音それぞれに対し，平均律からどの程度ピッチをずらしたいかをセント単位で指定する。

F0 7E Universal Non-Real Time SysEx header
<DevID> デバイスID（7F ＝ all devices）
08 sub-ID#1 ＝ MIDI Tuning Standard
08 sub-ID#2 ＝ Scale/Octave Tuning
ff 設定対象MIDIチャンネル指定用バイト列1
gg 設定対象MIDIチャンネル指定用バイト列2
hh 設定対象MIDIチャンネル指定用バイト列3
s1 C用チューニングパラメータ
s2 C#用チューニングパラメータ
s3 D用チューニングパラメータ
s4 D#用チューニングパラメータ
s5 E用チューニングパラメータ
s6 F用チューニングパラメータ
s7 F#用チューニングパラメータ
s8 G用チューニングパラメータ
s9 G#用チューニングパラメータ
s10 A用チューニングパラメータ
s11 A#用チューニングパラメータ
s12 B用チューニングパラメータ
F7 EOX

ff，gg，hhは，この設定をどのMIDIチャンネルに対して行うかを指定するものだ。それぞれのビットが各MIDIチャンネルに対応しており，対応するビットを1にするとそのチャンネルに設定が施される仕組みになっている。具体的には，

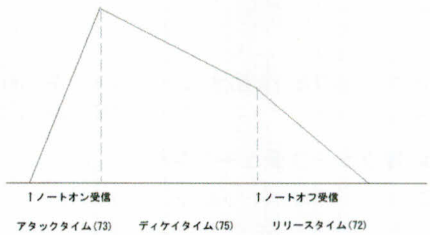
ffの第1ビット＝MIDIチャンネル16
ffの第0ビット＝MIDIチャンネル15
ggの第6ビット＝MIDIチャンネル14
ggの第5ビット＝MIDIチャンネル13
ggの第4ビット＝MIDIチャンネル12
ggの第3ビット＝MIDIチャンネル11
ggの第2ビット＝MIDIチャンネル10
ggの第1ビット＝MIDIチャンネル9
ggの第0ビット＝MIDIチャンネル8
hhの第6ビット＝MIDIチャンネル7
hhの第5ビット＝MIDIチャンネル6
hhの第4ビット＝MIDIチャンネル5
hhの第3ビット＝MIDIチャンネル4
hhの第2ビット＝MIDIチャンネル3
hhの第1ビット＝MIDIチャンネル2
hhの第0ビット＝MIDIチャンネル1

のようになっている。

s1～s12のチューニングパラメータは$00～$7fの範囲をとり，

$00　-64セント
$40　±0
$7f　+64セント

に対応する。



デフォルトである平均律は，s1～s12をすべて$40（±0）を設定したときと同等だ。

参考までに「純正律」と「アラビア音律」のチューニングパラメータを示しておこう。値の単位は「セント」だ。

	純正律（Cが主音）	アラビア音律
C	0	－6
C#	－8	+45
D	+4	－2
D#	+16	－12
E	－14	－51
F	－2	－8
F#	－10	+43
G	+2	－4
G#	+14	+47
A	－16	0
A#	+14	－10
B	－12	－49

たとえば，全GM2音源の全MIDIチャンネルをアラビア音律に指定する場合には，

F0 7E Universal Non-Real Time SysEx header
7F デバイスID（7F ＝ all devices）
08 sub-ID#1 ＝ MIDI Tuning Standard
08 sub-ID#2 ＝ Scale/Octave Tuning
03 設定対象MIDIチャンネル指定用バイト列1
7F 設定対象MIDIチャンネル指定用バイト列2
7F 設定対象MIDIチャンネル指定用バイト列3
3A C用チューニングパラメータ（-6）
6D C#用チューニングパラメータ（+45）
3E D用チューニングパラメータ（-2）
34 D#用チューニングパラメータ（-12）
0D E用チューニングパラメータ（-51）
38 F用チューニングパラメータ（-8）
6B F#用チューニングパラメータ（+43）
3C G用チューニングパラメータ（-4）
6F G#用チューニングパラメータ（+47）
40 A用チューニングパラメータ（0）
36 A#用チューニングパラメータ（-10）
0F B用チューニングパラメータ（-49）
F7 EOX

を送信すればいいことになる。

★GMシステムメッセージ

GM音源の初期化メッセージだが，GM2はGM1に対して上位互換を保っているものの，GM1から見るとGM2は別物ということができるため，初期化メッセージもこれまでのものがGM1専用とされ，GM2専用のものが新設された。

●GM2システムオン

F0 7E Universal Non-Real Time SysEx header
<DevID> デバイスID（7F ＝ all devices）
09 sub-ID#1 ＝ General MIDI message
03 sub-ID#2 ＝ General MIDI 2 On
F7 EOX

GM1の初期化は，sub-ID#2の部分が「01」だった。GM2の場合は「02」ではなく「03」である点に注意してほしい。

ところでGM2音源に対して「sub-ID#2＝01」を設定するとGM1レベルの初期化が行われ，逆にGM1音源に対して「03」を設定しても無視されてしまう。

●GMシステムオフ

GMシステムオフは，GM1のときと同じものが使える。

F0 7E Universal Non-Real Time SysEx header
<DevID> デバイスID（7F ＝ all devices）
09 sub-ID#1 ＝ General MIDI message
02 sub-ID#2 ＝ General MIDI Off
F7 EOX

★おわりに～ GM1∈GS，XG，GM1∈GM2 BUT GM2≠GS，XG

ユーザー側にとってやっかいなのは，今後出てくるGM2音源と現在持っているMIDI音源との非互換部分だ。

GM2音源はGM1音源との上位互換であるため，GM1音源用の音楽データなどは全て活用できる。が，GM2音源用の音楽データはGM1音源で正しく演奏することはできない。まぁ，これは下位互換性が保証されないから当たり前といえば当たり前だ。

報われないのは，GM1の拡張仕様であるローランドGS，ヤマハXGのフォーマットに対応した音源を所有するユーザーたちだ。両社共にGM1音源の上位互換なのだが，GM2とは異なる拡張を施しているためGM2とは基本的に互換性がないのだ。

「報われないかどうか」という問題は今後のGM2音源用のデータがどの程度増えてくるかにかかっているわけだが，とにかく，GM2が発表された今，MIDI音源の標準仕様がGM2になるのは目に見えているので，GM2発表以前のGS音源，XG音源のGM1互換機能は今後活用される機会はずいぶんと減るはずだ。

さて，気になるのはGS，XG陣営がGM2をどう迎えるか…という点だが，これは，特に「対抗する」ということもなく，GS，XGともにGM2を取り込む形で製品を発表していくようだ。

すでにローランドはGM2対応GS音源「SC8850」を発表しているし，ヤマハはすでに発売されていたXG音源「MU128」をGM2に対応させる無償アップデートプログラムを同社のWeb（http://www.yamaha.co.jp/product/decbx/news/gml2/index.html）で公開している。

あくまで「取り込む」というスタイルであり，今後もGS，XGともに独自に進化発展させていくようなので，結局ユーザー側からすると，音源を購入する際にはGM2，GS，XGという選択を迫られることになる。状況はGM1，GS，XGという三択からGM2，GS，XGという三択に変わっただけということだ。

ただ，表現力については，GM2になって大幅に高められたため，GSとXGとのポテンシャルの格差が縮まってはいる。よってローランドやヤマハ以外の楽器メーカーからかなり面白い音源が登場し，さらにそうしたGM2音源を活用したすごい表現力を持った演奏データなどが登場してくる可能性は極めて高い。「GS，XGに固執しないDTM界」がやってくる可能性はゼロではないのだ。

■コラム1

MIDIに関する最新情報が入手可能なサイト

MIDIに関する最新情報を入手する際には以下のサイトを訪れるといい。データ制作者はもちろん，MIDI関連のアプリケーション開発者は絶対にブックマークしておきたい。

●MMA（MIDI MANUFACTURERS ASSOCIATION）

米国のMIDI規格管理組織のサイト。MIDI規格に関する公式な情報が得られる。

URL＝http://www.midi.org/



●AMEI（社団法人 音楽電子事業協会）

日本の電子楽器関連メーカーが中心になって組織している団体。次期標準規格の立案やこれに関する調査研究を行っている。

URL＝http://www.amei.or.jp/



●AMEIのドキュメントのページ

MIDI規格に関する最新ドキュメントが閲覧できる。GM2に関する仕様書（英語版のみ）もここから入手可能だ。

URL=http://www.amei.or.jp/report/AMEI_MMA/documents.htm

●Harmony Central

ソフトウェアレビューから新モデルの紹介まで，MIDIに関する情報を総合的に取り扱う情報サイト。

URL＝http://www.harmonycentral.com/



■コラム2

GM2の煩雑な音色管理方法に戸惑う

私もGM2の仕様書を見て音色の管理方式に「バンク切り換え」が導入されるのは予想していた。14ビットあるバンク番号のうち下位バイトでバンクを切り換えるという手法も予測はしていたつもりだ。

しかし，バンク番号の上位バイトを属性バイトに流用する方式は私には予測できなかった。

元来美しいコード体系で名を馳せてきたMIDI規格もここにきて破綻してしまったのだろうか。

バンク切り換えの上位バイトは条件限定で別の用途に使う……というのは，あまりにも汚い。汚すぎる……。十万石まんじゅう……。

XGがリズムセットの音色バンクを似たような方式で管理していたり，GSがバンク番号の上位バイトのみをバンク切り換えに使用し，しかもそのバンク番号126，127をCM64系サウンドにマッピングし，さらに下のほうは48前後まで侵食されているあたりを踏まえ，中間的な裁定として現在の仕様を決めたのかもしれない。

しかし，そうして決めたGM2の，バンク管理を絡めた音色管理方式には，結局XG，GSとの互換性を絞り出すことができなかった。ならば，もうちょっと使いやすく，そしてユーザーにわかりやすいコード体系にすべきではなかったのだろうか。

代案としてもっとも単純なのは「パート属性を設定できる新しいコントロールチェンジ」を新設する方法だろう。これならば，以前に定めたバンク切り換えのコントロールチェンジの仕様をいじらずに新しいパラメータを設けることができる。

あるいは，ユニバーサルエクスクルーシブにそうしたパラメータを新設するのも悪くない。どのみちそうしたパート制御をするような上級ユーザーたちに「エクスクルーシブが難しい」という認識はないからだ。

とにかく，GM2の仕様のうち，ここだけはどうしても納得がいかないのであった。

リスト1 GM 2 Sound Set

PROG#	BANK#(MSB LSB)	GM2 TIMBRE NAME	Recommended Key Range
Piano			
1(00H)	79H 00H	Acoustic Grand Piano	21-108
	79H 01H	Acoustic Grand Piano (wide)	21-108
	79H 02H	Acoustic Grand Piano (dark)	21-108
2(01H)	79H 00H	Bright Acoustic Piano	21-108
	79H 01H	Bright Acoustic Piano (wide)	21-108
3(02H)	79H 00H	Electric Grand Piano	21-108
	79H 01H	Electric Grand Piano (wide)	21-108
4(03H)	79H 00H	Honky-tonk Piano	21-108
	79H 01H	Honky-tonk Piano (wide)	21-108
5(04H)	79H 00H	Electric Piano 1	28-103
	79H 01H	Detuned Electric Piano 1	28-103
	79H 02H	Electric Piano 1 (velocity mix)	28-103
	79H 03H	60's Electric Piano	28-103
6(05H)	79H 00H	Electric Piano 2	28-103
	79H 01H	Detuned Electric Piano 2	28-103
	79H 02H	Electric Piano 2 (velocity mix)	28-103
	79H 03H	EP Legend	28-103
	79H 04H	EP Phase	28-103
7(06H)	79H 00H	Harpsichord	41-89
	79H 01H	Harpsichord (octave mix)	41-89
	79H 02H	Harpsichord (wide)	41-89
	79H 03H	Harpsichord (with key off)	41-89
8(07H)	79H 00H	Clavi	36-96
	79H 01H	Pulse Clavi	36-96
Chromatic Percussion			
9(08H)	79H 00H	Celesta	60-108
10(09H)	79H 00H	Glockenspiel	72-108
11(0AH)	79H 00H	Music Box	60-84
12(0BH)	79H 00H	Vibraphone	53-89
	79H 01H	Vibraphone (wide)	53-89
13(0CH)	79H 00H	Marimba	48-84
	79H 01H	Marimba (wide)	48-84
14(0DH)	79H 00H	Xylophone	65-96
15(0EH)	79H 00H	Tubular Bells	60-77
	79H 01H	Church Bell	60-77
	79H 02H	Carillon	60-77
16(0FH)	79H 00H	Dulcimer	60-84
Organ			
17(10H)	79H 00H	Drawbar Organ	36-96
	79H 01H	Detuned Drawbar Organ	36-96
	79H 02H	Italian 60's Organ	36-96
	79H 03H	Drawbar Organ 2	36-96
18(11H)	79H 00H	Percussive Organ	36-96
	79H 01H	Detuned Percussive Organ	36-96
	79H 02H	Percussive Organ 2	36-96
19(12H)	79H 00H	Rock Organ	36-96
20(13H)	79H 00H	Church Organ	21-108
	79H 01H	Church Organ (octave mix)	21-108
	79H 02H	Detuned Church Organ	21-108
21(14H)	79H 00H	Reed Organ	36-96
	79H 01H	Puff Organ	36-96
22(15H)	79H 00H	Accordion	53-89
	79H 01H	Accordion 2	53-89
23(16H)	79H 00H	Harmonica	60-84
24(17H)	79H 00H	Tango Accordion	53-89
Guitar			
25(18H)	79H 00H	Acoustic Guitar (nylon)	40-84
	79H 01H	Ukulele	40-84
	79H 02H	Acoustic Guitar (nylon + key off)	40-84
	79H 03H	Acoustic Guitar (nylon 2)	40-84
26(19H)	79H 00H	Acoustic Guitar (steel)	40-84
	79H 01H	12-Strings Guitar	40-84
	79H 02H	Mandolin	40-84
	79H 03H	Steel Guitar with Body Sound	40-84
27(1AH)	79H 00H	Electric Guitar (jazz)	40-86
	79H 01H	Electric Guitar (pedal steel)	40-86
28(1BH)	79H 00H	Electric Guitar (clean)	40-86
	79H 01H	Electric Guitar (detuned clean)	40-86
	79H 02H	Mid Tone Guitar	40-86
29(1CH)	79H 00H	Electric Guitar (muted)	40-86
	79H 01H	Electric Guitar (funky cutting)	40-86
	79H 02H	Electric Guitar (muted velo-sw)	40-86
	79H 03H	Jazz Man	40-86
30(1DH)	79H 00H	Overdriven Guitar	40-86
	79H 01H	Guitar Pinch	40-86
31(1EH)	79H 00H	Distortion Guitar	40-86
	79H 01H	Distortion Guitar (with feedback)	40-86
	79H 02H	Distorted Rhythm Guitar	40-86
32(1FH)	79H 00H	Guitar Harmonics	40-86
	79H 01H	Guitar Feedback	40-86
Bass			
33(20H)	79H 00H	Acoustic Bass	28-55
34(21H)	79H 00H	Electric Bass (finger)	28-55
	79H 01H	Finger Slap Bass	28-55
35(22H)	79H 00H	Electric Bass (pick)	28-55
36(23H)	79H 00H	Fretless Bass	28-55
37(24H)	79H 00H	Slap Bass 1	28-55
38(25H)	79H 00H	Slap Bass 2	28-55
39(26H)	79H 00H	Synth Bass 1	28-55
	79H 01H	Synth Bass (warm)	28-55
	79H 02H	Synth Bass 3 (resonance)	28-55
	79H 03H	Clavi Bass	28-55
	79H 04H	Hammer	28-55
40(27H)	79H 00H	Synth Bass 2	28-55
	79H 01H	Synth Bass 4 (attack)	28-55
	79H 02H	Synth Bass (rubber)	28-55
	79H 03H	Attack Pulse	28-55
Strings&Orchestral instruments			
41(28H)	79H 00H	Violin	55-96
	79H 01H	Violin (slow attack)	55-96
42(29H)	79H 00H	Viola	48-84
43(2AH)	79H 00H	Cello	36-72
44(2BH)	79H 00H	Contrabass	28-55
45(2CH)	79H 00H	Tremolo Strings	28-96
46(2DH)	79H 00H	Pizzicato Strings	28-96
47(2EH)	79H 00H	Orchestral Harp	23-103
	79H 01H	Yang Chin	23-103
48(2FH)	79H 00H	Timpani	36-57
Ensemble			
49(30H)	79H 00H	String Ensembles 1	28-96
	79H 01H	Strings and Brass	28-96
	79H 02H	60s Strings	28-96
50(31H)	79H 00H	String Ensembles 2	28-96
51(32H)	79H 00H	SynthStrings 1	36-96
	79H 01H	SynthStrings 3	36-96
52(33H)	79H 00H	SynthStrings 2	36-96
53(34H)	79H 00H	Choir Aahs	48-79
	79H 01H	Choir Aahs 2	48-79
54(35H) 7	9H 00H	Voice Oohs	48-79
	79H 01H	Humming	48-79
55(36H)	79H 00H	Synth Voice	48-84
	79H 01H	Analog Voice	48-84
56(37H)	79H 00H	Orchestra Hit	48-72
	79H 01H	Bass Hit Plus	48-72
	79H 02H	6th Hit	48-72
	79H 03H	Euro Hit	48-72
Brass			
57(38H)	79H 00H	Trumpet	58-94
	79H 01H	Dark Trumpet Soft	58-94
58(39H)	79H 00H	Trombone	34-75
	79H 01H	Trombone 2	34-75
	79H 02H	Bright Trombone	34-75
59(3AH)	79H 00H	Tuba	29-55
60(3BH)	79H 00H	Muted Trumpet	58-82
	79H 01H	Muted Trumpet 2	58-82
61(3CH)	79H 00H	French Horn	41-77
	79H 01H	French Horn 2 (warm)	41-77
62(3DH)	79H 00H	Brass Section	36-96
	79H 01H	Brass Section 2 (octave mix)	36-96
63(3EH)	79H 00H	Synth Brass 1	36-96
	79H 01H	Synth Brass 3	36-96
	79H 02H	Analog Synth Brass 1	36-96
	79H 03H	Jump Brass	36-96
64(3FH)	79H 00H	Synth Brass 2	36-96
	79H 01H	Synth Brass 4	36-96
	79H 02H	Analog Synth Brass 2	36-96
Reed			
65(40H)	79H 00H	Soprano Sax	54-87
66(41H)	79H 00H	Alto Sax	49-80
67(42H)	79H 00H	Tenor Sax	42-75
68(43H)	79H 00H	Baritone Sax	37-68
69(44H)	79H 00H	Oboe	58-91
70(45H)	79H 00H	English Horn	52-81
71(46H)	79H 00H	Bassoon	34-72
72(47H)	79H 00H	Clarinet	50-91
Pipe			
73(48H)	79H 00H	Piccolo	74-108
74(49H)	79H 00H	Flute	60-96
75(4AH)	79H 00H	Recorder	60-96
76(4BH)	79H 00H	Pan Flute	60-96
77(4CH)	79H 00H	Blown Bottle	60-96
78(4DH)	79H 00H	Shakuhachi	55-84
79(4EH)	79H 00H	Whistle	60-96
80(4FH)	79H 00H	Ocarina	60-84
Synth Lead			
81(50H)	79H 00H	Lead 1 (square)	21-108
	79H 01H	Lead 1a (square 2)	21-108
	79H 02H	Lead 1b (sine)	21-108
82(51H)	79H 00H	Lead 2 (sawtooth)	21-108
	79H 01H	Lead 2a (sawtooth 2)	21-108
	79H 02H	Lead 2b (saw + pulse)	21-108
	79H 03H	Lead 2c (double sawtooth)	21-108
	79H 04H	Lead 2d (sequenced analog)	21-108
83(52H)	79H 00H	Lead 3 (calliope)	36-96
84(53H)	79H 00H	Lead 4 (chiff)	36-96
85(54H)	79H 00H	Lead 5 (charang)	36-96
	79H 01H	Lead 5a (wire lead)	36-96
86(55H)	79H 00H	Lead 6 (voice)	36-96
87(56H)	79H 00H	Lead 7 (fifths)	36-96
88(57H)	79H 00H	Lead 8 (bass + lead)	21-108
	79H 01H	Lead 8a (soft wrl)	21-108
Synth Pad			
89(58H)	79H 00H	Pad 1 (new age)	36-96
90(59H)	79H 00H	Pad 2 (warm)	36-96
	79H 01H	Pad 2a (sine pad)	36-96
91(5AH)	79H 00H	Pad 3 (polysynth)	36-96
92(5BH)	79H 00H	Pad 4 (choir)	36-96
	79H 01H	Pad 4a (itopia)	36-96
93(5CH)	79H 00H	Pad 5 (bowed)	36-96
94(5DH)	79H 00H	Pad 6 (metallic)	36-96
95(5EH)	79H 00H	Pad 7 (halo)	36-96
96(5FH)	79H 00H	Pad 8 (sweep)	36-96
Synth SFX			
97(60H)	79H 00H	FX 1 (rain)	36-96
98(61H)	79H 00H	FX 2 (soundtrack)	36-96
99(62H)	79H 00H	FX 3 (crystal)	36-96
	79H 01H	FX 3a (synth mallet)	36-96
100(63H)	79H 00H	FX 4 (atmosphere)	36-96
101(64H)	79H 00H	FX 5 (brightness)	36-96
102(65H)	79H 00H	FX 6 (goblins)	36-96
103(66H)	79H 00H	FX 7 (echoes)	36-96
	79H 01H	FX 7a (echo bell)	36-96
	79H 02H	FX 7b (echo pan)	36-96
104(67H)	79H 00H	FX 8 (sci-fi)	36-96
Ethnic Misc.			
105(68H)	79H 00H	Sitar	48-77
	79H 01H	Sitar 2 (bend)	48-77
106(69H)	79H 00H	Banjo	48-84
107(6AH)	79H 00H	Shamisen	50-79
108(6BH)	79H 00H	Koto	55-84
	79H 01H	Taisho Koto	55-84
109(6CH)	79H 00H	Kalimba	48-79
110(6DH)	79H 00H	Bag pipe	36-77
111(6EH)	79H 00H	Fiddle	55-96
112(6FH)	79H 00H	Shanai	48-72
Percussive			
113(70H)	79H 00H	Tinkle Bell	72-84
114(71H)	79H 00H	Agogo	60-72
115(72H)	79H 00H	Steel Drums	52-76
116(73H)	79H 00H	Woodblock	*
	79H 01H	Castanets	*
117(74H)	79H 00H	Taiko Drum	*
	79H 01H	Concert Bass Drum	*
118(75H)	79H 00H	Melodic Tom	*
	79H 01H	Melodic Tom 2 (power)	*
119(76H)	79H 00H	Synth Drum	*
	79H 01H	Rhythm Box Tom	*
	79H 02H	Electric Drum	*
120(77H)	79H 00H	Reverse Cymbal	*
SFX			
121(78H)	79H 00H	Guitar Fret Noise	*
	79H 01H	Guitar Cutting Noise	*
	79H 02H	Acoustic Bass String Slap	*
122(79H)	79H 00H	Breath Noise	*

	79H 01H	Flute Key Click	*
123(7AH)	79H 00H	Seashore	*
	79H 01H	Rain	*
	79H 02H	Thunder	*
	79H 03H	Wind	*
	79H 04H	Stream	*
	79H 05H	Bubble	*
124(7BH)	79H 00H	Bird Tweet	*
	79H 01H	Dog	*
	79H 02H	Horse Gallop	*
	79H 03H	Bird Tweet 2	*
125(7CH)	79H 00H	Telephone Ring	*
	79H 01H	Telephone Ring 2	*
	79H 02H	Door Creaking	*
	79H 03H	Door	*
	79H 04H	Scratch	*
	79H 05H	Wind Chime	*
126(7DH)	79H 00H	Helicopter	*
	79H 01H	Car Engine	*
	79H 02H	Car Stop	*
	79H 03H	Car Pass	*
	79H 04H	Car Crash	*
	79H 05H	Siren	*
	79H 06H	Train	*
	79H 07H	Jetplane	*
	79H 08H	Starship	*
	79H 09H	Burst Noise	*
127(7EH)	79H 00H	Applause	*
	79H 01H	Laughing	*
	79H 02H	Screaming	*
	79H 03H	Punch	*
	79H 04H	Heart Beat	*
	79H 05H	Footsteps	*
128(7FH)	79H 00H	Gunshot	*
	79H 01H	Machine Gun	*
	79H 02H	Lasergun	*
	79H 03H	Explosion	*

リスト2 GM 2 Percussion Sound Set

	PC#1 STANDARD		PC#9 Set ROOM		PC#17 Set POWER Set	
NOTE#	Inst.Name	PAN	Inst.Name	PAN	Inst.Name	PAN
27 (D#)	High Q	49	@		@	
28 (E)	Slap	49	@		@	
29 (F)	Scratch Push [EXC7]	54	@		@	
30 (F#)	Scratch Pull [EXC7]	54	@		@	
31 (G)	Sticks	64	@		@	
32 (G#)	Square Click	54	@		@	
33 (A)	Metronome Click	64	@		@	
34 (A#)	Metronome Bell	64	@		@	
35 (B)	Acoustic Bass Drum	64	@		@	
36 (C)	Bass Drum 1	64	@		Power Kick Drum	64
37 (C#)	Side Stick	64	@		@	
38 (D)	Acoustic Snare	64	@		Power Snare Drum	64
39 (D#)	Hand Clap	54	@		@	
40 (E)	Electric Snare	64	@		@	
41 (F)	Low Floor Tom	34	Room Low Tom 2	34	Power Low Tom 2	34
42 (F#)	Closed Hi-hat [EXC1]	84	@		@	
43 (G)	High Floor Tom	46	Room Low Tom 1	46	Power Low Tom 1	46
44 (G#)	Pedal Hi-hat [EXC1]	84	@		@	
45 (A)	Low Tom	58	Room Mid Tom 2	58	Power Mid Tom 2	58
46 (A#)	Open Hi-hat [EXC1]	84	@		@	
47 (B)	Low-Mid Tom	70	Room Mid Tom 1	70	Power Mid Tom 1	70
48 (C)	High Mid Tom	82	Room Hi Tom 2	82	Power Hi Tom 2	82
49 (C#)	Crash Cymbal 1	84	@		@	
50 (D)	High Tom	94	Room Hi Tom 1	94	Power Hi Tom 1	94
51 (D#)	Ride Cymbal 1	44	@		@	
52 (E)	Chinese Cymbal	44	@		@	
53 (F)	Ride Bell	44	@		@	
54 (F#)	Tambourine	74	@		@	
55 (G)	Splash Cymbal	54	@		@	
56 (G#)	Cowbell	84	@		@	
57 (A)	Crash Cymbal 2	44	@		@	
58 (A#)	Vibra-slap	29	@		@	
59 (B)	Ride Cymbal 2	44	@		@	
60 (MID C)	High Bongo	99	@		@	
61 (C#)	Low Bongo	99	@		@	
62 (D)	Mute Hi Conga	39	@		@	
63 (D#)	Open Hi Conga	39	@		@	
64 (E)	Low Conga	44	@		@	
65 (F)	High Timbale	84	@		@	
66 (F#)	Low Timbale	84	@		@	
67 (G)	High Agogo	29	@		@	
68 (G#)	Low Agogo	29	@		@	
69 (A)	Cabasa	29	@		@	
70 (A#)	Maracas	24	@		@	
71 (B)	Short Whistle [EXC2]	99	@		@	
72 (C)	Long Whistle [EXC2]	99	@		@	
73 (C#)	Short Guiro [EXC3]	94	@		@	
74 (D)	Long Guiro [EXC3]	94	@		@	
75 (D#)	Claves	84	@		@	
76 (E)	Hi Wood Block	99	@		@	
77 (F)	Low Wood Block	99	@		@	
78 (F#)	Mute Cuica [EXC4]	44	@		@	
79 (G)	Open Cuica [EXC4]	44	@		@	
80 (G#)	Mute Triangle [EXC5]	24	@		@	
81 (A)	Open Triangle [EXC5]	24	@		@	
82 (A#)	Shaker	94	@		@	
83 (B)	Jingle Bell	99	@		@	
84 (C)	Bell Tree	104	@		@	
85 (C#)	Castanets	34	@		@	
86 (D)	Mute Surdo [EXC6]	44	@		@	
87 (D#)	Open Surdo [EXC6]	44	@		@	
88 (E)	--- --- ---					

	PC#25 ELECTRONIC Set		PC#26 ANALOG Set		PC#33 JAZZ Set	
NOTE#	Inst.Name	PAN	Inst.Name	PAN	Inst.Name	PAN
27 (D#)	@		@		@	
28 (E)	@		@		@	
29 (F)	@		@		@	
30 (F#)	@		@		@	
31 (G)	@		@		@	
32 (G#)	@		@		@	
33 (A)	@		@		@	
34 (A#)	@		@		@	
35 (B)	@		@		Jazz Kick 2	64
36 (C)	Electric Bass Drum	64	Analog Bass Drum	64	Jazz Kick 1	64
37 (C#)	@		Analog Rim Shot	64	@	
38 (D)	Electric Snare 1	64	Analog Snare 1	64	@	
39 (D#)	@		@		@	
40 (E)	Electric Snare 2	64	@		@	
41 (F)	Electric Low Tom 2	34	Analog Low Tom 2	34	@	
42 (F#)	@		Analog CHH 1 [EXC1]	84	@	
43 (G)	Electric Low Tom 1	46	Analog Low Tom 1	46	@	
44 (G#)	@		Analog CHH 2 [EXC1]	84	@	
45 (A)	Electric Mid Tom 2	58	Analog Mid Tom 2	58	@	
46 (A#)	@		Analog OHH [EXC1]	84	@	
47 (B)	Electric Mid Tom 1	70	Analog Mid Tom 1	70	@	
48 (C)	Electric Hi Tom 2	82	Analog Hi Tom 2	82	@	
49 (C#)	@		Analog Cymbal	84	@	
50 (D)	Electric Hi Tom 1	94	Analog Hi Tom 1	94	@	
51 (D#)	@		@		@	
52 (E)	Reverse Cymbal 44		@		@	
53 (F)	@		@		@	
54 (F#)	@		@		@	
55 (G)	@		@		@	
56 (G#)	@		Analog Cowbell	84	@	
57 (A)	@		@		@	
58 (A#)	@		@		@	
59 (B)	@		@		@	
60 (MID C)	@		@		@	
61 (C#)	@		@		@	
62 (D)	@		Analog High Conga	39	@	
63 (D#)	@		Analog Mid Conga	44	@	
64 (E)	@		Analog Low Conga	49	@	
65 (F)	@		@		@	
66 (F#)	@		@		@	
67 (G)	@		@		@	
68 (G#)	@		@		@	
69 (A)	@		@		@	
70 (A#)	@		Analog Maracas	24	@	
71 (B)	@		@		@	
72 (C)	@		@		@	
73 (C#)	@		@		@	
74 (D)	@		@		@	
75 (D#)	@		Analog Claves	84	@	
76 (E)	@		@		@	
77 (F)	@		@		@	
78 (F#)	@		@		@	
79 (G)	@		@		@	
80 (G#)	@		@		@	
81 (A)	@		@		@	
82 (A#)	@		@		@	
83 (B)	@		@		@	
84 (C)	@		@		@	
85 (C#)	@		@		@	
86 (D)	@		@		@	
87 (D#)	@		@		@	
88 (E)	---		---		---	

	PC#41 BRUSH Set		PC#49 ORCHESTRA Set		PC#57 SFX Set	
NOTE#	Inst.Name	PAN	Inst.Name	PAN	Inst.Name	PAN
27 (D#)	@		Closed Hi-hat 2[EXC1]	84	---	
28 (E)	@		Pedal Hi-hat [EXC1]	84	---	
29 (F)	@		Open Hi-hat 2 [EXC1]	84	---	
30 (F#)	@		Ride Cymbal 1	44	---	
31 (G)	@		@		---	
32 (G#)	@		@		---	
33 (A)	@		@		---	
34 (A#)	@		@		---	
35 (B)	Jazz Kick 2	64	Concert BD 2	24	---	
36 (C)	Jazz Kick 1	64	Concert BD 1	24	---	
37 (C#)	@		@		---	
38 (D)	Brush Tap	64	Concert SD	44	---	
39 (D#)	Brush Slap	64	Castanets	34	High Q	49
40 (E)	Brush Swirl	64	Concert SD	44	Slap	49
41 (F)	@		Timpani F	34	Scratch Push [EXC7]	54
42 (F#)	@		Timpani F#	34	Scratch Pull [EXC7]	54
43 (G)	@		Timpani G	34	Sticks	64
44 (G#)	@		Timpani G#	34	Square Click	54
45 (A)	@		Timpani A	34	Metronome Click	64
46 (A#)	@		Timpani A#	34	Metronome Bell	64
47 (B)	@		Timpani B	34	Guitar Fret Noise	64
48 (C)	@		Timpani c	34	Guitar Cutting Noise Up	
64						
49 (C#)	@		Timpani c#	34	Guitar Cutting Noise	
Down	64					
50 (D)	@		Timpani d	34	String Slap of Double	
Bass	64					
51 (D#)	@		Timpani d#	34	Fl.Key Click	64
52 (E)	@		Timpani e	34	Laughing	64
53 (F)	@		Timpani f	34	Scream	64
54 (F#)	@		@		Punch	64
55 (G)	@		@		Heart Beat	64
56 (G#)	@		@		Footsteps 1	64
57 (A)	@		Concert Cymbal 2	34	Footsteps 2	64
58 (A#)	@		@		Applause	64
59 (B)	@		Concert Cymbal 1	34	Door Creaking	64
60 (MID C)	@		@		Door	64
61 (C#)	@		@		Scratch	64
62 (D)	@		@		Wind Chimes	64
63 (D#)	@		@		Car-Engine	64
64 (E)	@		@		Car-Stop	64
65 (F)	@		@		Car-Pass	64
66 (F#)	@		@		Car-Crash	64
67 (G)	@		@		Siren	64
68 (G#)	@		@		Train	64
69 (A)	@		@		Jetplane	64
70 (A#)	@		@		Helicopter	64
71 (B)	@		@		Starship	64
72 (C)	@		@		Gun Shot	64
73 (C#)	@		@		Machine Gun	64
74 (D)	@		@		Lasergun	64
75 (D#)	@		@		Explosion	64
76 (E)	@		@		Dog	64
77 (F)	@		@		Horse-Gallop	64
78 (F#)	@		@		Birds	64
79 (G)	@		@		Rain	64
80 (G#)	@		@		Thunder	64
81 (A)	@		@		Wind	64
82 (A#)	@		@		Seashore	64
83 (B)	@		@		Stream	64
84 (C)	@		@		Bubble	64
85 (C#)	@		@		---	
86 (D)	@		@		---	
87 (D#)	@		@		---	
88 (E)	---					

Notes: ---:Does not sound
@ :Use Standard Set Instrument
[EXC]:Instruments which have same EXC numbers do not sound simultaneously.
Program Change messages in consumer documentation are normally one-based; therefore, the decimal value of PC #1 presented here (Standard Set) is equivalent to 00H.

特別企画 西暦2000年のMIDI環境

GM2音源SC-8850を聞く

松尾直樹　Matsuo Naoki

GM1時代のSC-55の再来となるか？　新世代GM2音源のリファレンスともいえるのがこのローランドのSC-8850だ。ハイスペック標準音源というGM2音源の可能性を垣間見てみたい。

SC-8850登場

SC-88Proが登場して早3年，ライバル各社が最大同時発音数128音，パート数64チャンネル，GM2対応，1600音色を超える音色数を誇るスーパー音源を発表するなか，御大ローランドが重い腰を上げて投入したのがSC-8850です。ローランドはもうDTM分野から離れていくんじゃないかという悪い噂も耳にしたのですが，現在のDTMを創製してきたといっても過言ではないローランド，好評を博したSC-88Proをそのままに，さらによりアグレッシブな音源としてのSC-8850を発表しました。当時(1999年夏)最新のGeneralMIDI2に対応した初の製品ということでも注目を集めていました。ここでは，GM2時代の標準音源ともいえる同機の試用レポートをしてみたいと思います。

外観は，それまでのSCシリーズの高級感のある黒から銀色がかったものに変わり，DATデッキやビデオデッキとも調和しそうな感じです。またSC-88Proで懸念されていた操作性も，大型液晶ディスプレイとVALUEダイヤル，見直されたボタン配置のおかげで格段によくなっています。これまで難儀していた音色の選択もダイヤルを回すだけで瞬時に変更することができるわけで，この便利さに慣れるとSC-88Proの操作性がいかに悪いか考えさせられます。

また，これまで前面にあったMIDI IN B端子が後ろに回ったことで，よりデザインがよくなったような印象があります。背面にはさらに，最近流行のUSB端子も搭載されました。これまでのMIDIやシリアルによるパソコンとの接続に加え，USB接続も可能です。MIDIでの接続では32パートまでしか使えないので，SC-8850の64パートを同時に使うにはこのUSB端子での接続かシリアルでの接続が必要になるのですが，まだまだUSBは不安定のようです。しかし，パソコンとキーボードを繋ぐケーブルの間にHDDや音源が繋がっている様子は，なにか隔世の感があります。凄い時代になりました。

電源ONで音を聞いてみる

まず電源を入れると目に留まるのが，音色名ごとに用意された楽器のドット絵です。VALUEダイヤルを回すたびに変化するので，いままで文字だけでしかイメージできなかった楽器の姿をよりグラフィカルに確認することができます。また，膨大な音色の確認法としてボリュームボタンを押さえることによるプレビューがありますが，これまで無機質な単音のみだったものに，楽器ごとに用意された，その楽器の美味しいフレーズプレイが追加されました。ベンドやベロシティなどの変化も豊富で，これを聞くだけでも楽しめます。実際私はこのプレビュープレイだけで何時間も費やしてしまったのですが，膨大な音の変化や液晶ドット変化と相まって，そんな時間や苦痛をまったく感じさせないほど楽しいものになりました。この力の入れ方は，音源の高性能化だけにとどまらないローランドの製品開発に対するこだわりを感じます。

追加された音色たち

SC-88Proに比べると，今回の音色追加はかなり偏ったように感じます。というのも，ある楽器音色はほとんどが新しい音色としてSC-8850マップに配置されている場合もあれば，なかにはSC-88Proマップとすべて共通という場合もあります。前者は，SC-88Proの追加音色として少なかった部分を補った場合が多いのですが，顕著な追加音色としてはピアノ，クラビネット，ハーモニカ，シンセベース，ハープ，ストリングス，コーラス，シンセリード，パーカッション，SFXです。特にシンセやストリングスは，SC-8850の同時発音数128音を生かし，これまで最大でも2パーシャルを使う音色だけでしたが，3パーシャル，4パーシャルを使う音色が多数追加され，非常に重厚で豊かなサウンドになっています。

また，今回追加された音色でもっとも力が入れられているのが#54のボイス系コーラスで，JAZZ系の「だっだだっだ?」や「ふふふ?」のようなハミング風ボーカルならSC-8850だけでも作れそうです。

そのほか，ちょっと一般的な使用を考えると首を傾げたくなるのですが，アジアの民族音楽やSFXが多数追加されています。特にノイズ系のSFXが多数追加されているのですが，これほど必要なのかどうかは非常に疑問です。これも1音色と考えると単なる音色数稼ぎのような気がしてしまうのですが……。

その半面まったくといっていいほど追加されていない音色はギターとベースです。個人的には毎回注目している音色なのですが，ほとんどSC-88Proマップと共通音色です。サックスやフルートなどの管楽器も，これといった追加はありません。

サンプリング音源としてこの分野でできることは，とりあえずSC-88Proでやり尽くしたということなのかもしれませんが，もう少し奮起してもらいたかったところです。

ドラムセットは？

ドラムセットに目を移せば，Standard1やROOM，JAZZといった基本セットはほとんどの音がサンプリングし直されているようで，よりそれぞれのセット別キャラクターがはっきりしてきたように思います。また同じ種類のセットでも，Standard L/RやJAZZ L/Rといった，よりステレオの音場感を強くした(?)音色が追加されています。早い話がハイハットやスネアなどの音がデフォルトでランダムパンポットにしてあるだけなんですけど，ハイハットやタムが左右あちこちで飛び交うような幻想的(?)効果がお手軽に出せます。しかし，このランダムパンポットはデータで指定してしまえばSC-55なんかでも可能なわけで，あたかもSC-8850の新しい音色のようにわざわざ扱う必要があるのかどうか疑問に思います。どうせなら純粋な音色追加をお願いしたいところです。

また，リズムマシンメーカーとして定評のあるローランドにしては，ダンスやテクノ方面といった音色追加に目新しさが全然ないというのも寂しいところです。もっとも，SC-88Proの時点である程度やることはやってしまった感も否めませんが……。

今回の音色追加のテーマが民族楽器やSFXの充実にあるのかどうかはわかりませんが，ドラムセットもやはりガムランやSFX系の追加音色が目立ちます。一般的利用を考えると疑問ですが，まあ，最近流行のDTV(デスクトップビデオ)作品の効果音作成といった用途にもSC-8850が活躍できるかも





しれません。安価な効果音を集めたようなCDはもう必要ないくらい多数追加されています。

気になる互換性

さて，もっとも既存のSC-88Proユーザーが気にしているであろう部分，SC-8850はSC-88Proをリプレースできるかどうかです。SC-88Proを下取りにしてSC-8850を買い換えても過去のデータがまったく同じように鳴るのかどうか。また，今後増えていくであろうSC-8850ユーザーの人たちが，自分が作った旧音源用データを同じように聞いてもらえるかどうか。データ制作者なら皆さん考えることだろうと思います。

私が手持ちのSC-88Proデータを再生させた感じでは，多少音の抜けがよくなっているような印象はありますが，ほぼ同じように鳴っているようです。ただ，フィルタ，特にアタックタイムやレゾナンス，カットオフフリーケンシーを操作しているデータは，少し違った印象に聞こえます。SC-8850のフィルタ変化のほうがやや激しいようです。また，顕著な違いでいえば，ドラムのクローズドハイハットがかなり違います。やや開いた感じといいましょうか。文字だと伝えにくいですが「チッチチ」が「スッスス」(ここまで極端ではないですが……)に変わった感じです。ベロシティの感度も変わっているように思います。

この違いを大きいととるか小さいととるかは意見のわかれるところだと思いますが，旧音源との完全上位互換でないことは知っておくべきではないでしょうか。SC-88からSC-88Proへの移行が問題なかっただけに私自身も同様の期待をしていたのですが，ちょっと残念でした。もちろん完全な互換がないといっても多少音の印象が変わるだけで，まったく別の音が鳴ってしまったり，エラーが続発して再生できないと言うものでもありませんので，広告に偽りなし……ということになるとは思いますが……。

データ操作の変更は？

データ面での操作性はこれまでとほとんど変わりませんので，旧音源を扱い慣れている人ならば簡単にSC-8850の機能を使いこなしたデータを作ることができるかと思います。

エクスクルーシブ命令もGM2Setなどのコマンドが増えた程度で大きな変更はありませんし，音色マップの切り替えも従来と同様，コントロールチェンジ32番のバンクセレクト下位バイトで指定するだけです。

Z-MUSICでいうなら，i0,2@1でSC-88MAPのピアノ音色指定ですが，iコマンドの2を4，つまりi0,4@1と変更するだけでSC-8850のピアノ音色に変わります。

エフェクトは従来どおり

意外にも今回まったく手が加えられていないのがエフェクト面です。私の勝手な予想でインサーションエフェクトは3つまで同時使用できるようになっているのではないかと思っていたのですが，従来通どおりひとつまでで，EQに関しても全体に対する指定のみです。もっとも，ここに手を加えるとMU128と同じになってしまうわけで，そこは御大ローランド，独自のポリシーを持っての判断かと思います。

まあ，エフェクトをかけなくてもそれぞれの音色で十分よい音を出していますし，エフェクトを高機能にしてもエントリーレベルの人には無用なものかもしれません。そこで値段が上がってしまったり，肝心の音色のほうがおろそかになっても困りものです。そう考えると，エフェクトに関しての性能アップを見送ったのは正しい判断ではないでしょうか。

買いかどうか

SC-8850の操作性や新しい音色，発音数等に魅力を感じる人ならば迷わず購入でしょう。SC-88Pro登場時と同じ値段で，ここまで高機能になるのですから時代の進歩を感じます。

ただし前述のように，過去のデータ資産がまったく同じように再生できるわけではありませんのでその辺が気になる人は迷うところでしょう。しかし，SC-8850登場によって旧音源の中古相場が下落する可能性もありますので，お金に余裕のある人は両方手に入れるのも手だと思います。

ただここ数年，MP3をはじめとする音声圧縮技術の進歩やISDNの普及，パソコン通信からインターネットへのシフト，今後は通信を介してMIDIデータを配布し，1台の音源でいかに音楽を演奏するか……というプラモデルや箱庭的音楽制作は次第に廃れていくように思います。この価格帯も微妙なところで，もう数万円加えればプロ仕様の音源が手に入ります。複数音源を駆使した音楽制作や音作りを考えるなら，そちらに食指を動かすのも手でしょう。

もちろん，SC-8850はマルチティンバー音源としての最高峰に位置しますし，音質も素晴らしく，プロの現場でも十分すぎるくらいの活躍をすることになるでしょう。あとは，皆さんがどのような音楽を目指すかにかかってくると思います。膨大な音色数を前に，新たな作品への創造力を喚起させられる人も多いでしょう。音源という素材は素晴らしいものsが手に入る世の中になりました。問題はこれをどう活用するかです。

Oh!Xでは皆さまからの音楽作品をお待ちしています。

主な特徴

64パート，同時発音数128ボイス
1640音色+63ドラムセット
USBインタフェイス搭載

主な仕様

●パート数：64
●最大同時発音数：128音（ボイス）
●本体メモリ：音色マップ=4（SC-8850，SC-88Pro，SC-88，SC-55），プリセット音色数=1640，ドラム音色セット=63，ユーザー音色数=256，ユーザードラム音色セット=2
●エフェクト：リバーブ（8種類），コーラス（8種類），ディレイ（10種類），2バンドイコライザー，インサーションエフェクト（64種類）
●ディスプレイ：160×64ドット/グラフィックLCD（バック照明付き）
●接続端子：MIDIコネクタ（IN1，IN2，OUT1，OUT2），オーディオ・インプットジャック（ステレオ），オーディオアウトプット1ジャック（ステレオ），オーディオアウトプット2ジャック（ステレオ），ヘッドホンジャック，シリアルコネクタ，USBコネクタ
●電源：AC100V（50/60Hz）
●消費電力：11W
最大外形寸法：218（W）×278（D）×88（H）mm
●重量：2.3kg
●付属品：取扱説明書，保証書，電源ケーブル
●別売品：コンピュータケーブル（RSC-15N=NEC PC-9800シリーズ用，RSC-15AT=IBM PC/ATシリーズ用，RSC-15APL=Apple Macintosh用
※PC-9800シリーズのシリアル接続には対応していません

WindowsでZ-MUSICシステムを使う

須賀真 Suga Makoto
ZMC3.EXE/Z2M3.EXE 八木康成 Yagi Yasunari
ZEDIT.EXE 佐原政治(MASA) Sahara Masaharu

X68000ではお馴染みのMMLによるミュージックデータ制作。Z-MUSICシステムは，簡潔な記述と非常にコンパクトなデータで多彩な表現が可能で，音源のポテンシャルをフルに引き出すことができます。この環境をWindowsでも使えるようにするプロジェクトのひとつを紹介します。

★最近のDTM事情

旧Oh!X休刊の頃はDTMといえばローランドSC-55/88全盛期でした。X68000においてはZ-MUSICによるデータが多く作られ，Oh!X誌上はもちろん，草の根ネットなどでもハイレベルな作品が次々と発表されていました。

あれから約4年，ローランドはSC-88Pro，SC-8850と超弩級スペックの音源を発売，新たな勢力としてヤマハのMUシリーズも台頭してきました。これらの音源の表現力は並々ならぬものがあります。そしてZ-MUSICもさらに高度な表現が可能になったVer.3が発表され，ますます盛り上がりを見せるかと思われました。……が。

その頃からX68000ユーザーもWindowsマシンに乗り換える傾向が見られ，X68000の衰退とともにZ-MUSICのデータもあまり作られなくなっていきました。

私もいつのまにか普段はWindowsをメインに使うようになってきましたが，音楽制作だけはZ-MUSICから離れられずにいました。どうもMMLに慣れてしまうと一般のシーケンサは使いづらいですね。お値段も結構しますし。

ところが最新の音源では売りである64パートを使うにはUSB接続や専用ドライバが必要などX68000には厳しい状況になりつつあります。次機種にはMIDI端子がついているという保証はどこにもありません。

それに，あまり考えたくないですがX68000がいつまで持つのかという問題もありますね。そろそろX68000から離れたところでのMIDIデータ制作環境も考えなくてはなりません。

★Z-MUSICって？

昔からのX68000ユーザーの方には説明は不要と思いますが，Z-MUSICとは本誌でもお馴染みの西川善司氏の開発されたX68000用の音楽システムで，X68000の内蔵FM音源，AD PCM，外部MIDI機器を制御するドライバとMMLによる音楽データ制作環境などで構成されています。

MMLとは「Music Macro Language」，すなわち音楽記述言語です。MMLによるデータ制作では一般のシーケンサのように楽譜やピアノロールのようなグラフィカルなエディット画面は使いません。たとえば演奏部分では「ド」を4分音符で鳴らしたければ"C4"とテキストファイルに記述し，それをコンパイルして演奏ファイルを作るのが一般的な方法です。

この手順はC言語などの開発言語にも似ていますね。プログラミングの経験がある方や，HTMLを書くのに専用エディタよりメモ帳のほうが使いやすいという方には馴染みやすいのではないでしょうか。

取っ付きにくいけど，慣れてしまえば高速かつ高度な入力ができるというのがMML方式の長所です。半面，微妙なニュアンスを表現するための細かいコントロールなどは入力が大変です。

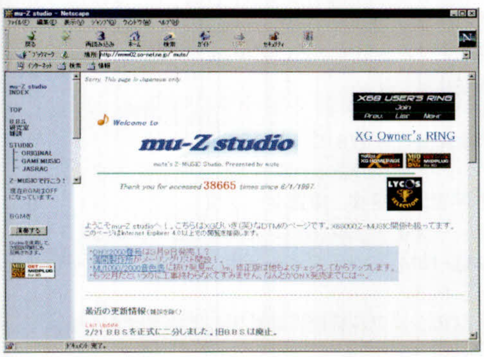
おそらく世界最強と思われるMML体系を持つZ-MUSICはMML方式の苦手とする和音やポルタメントなどの表記を簡潔に書くことができたり，ARCCというシステムで連続的なコントロール変化の記述ができたりと，非常によくできています。

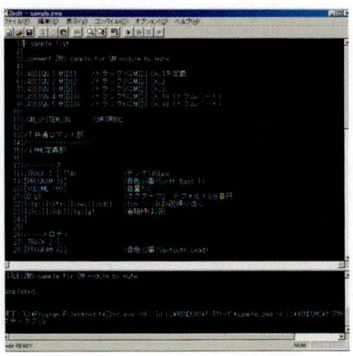
まさに慣れてしまえばほかのシーケンサには乗り換えられない，麻薬的なシステムです。

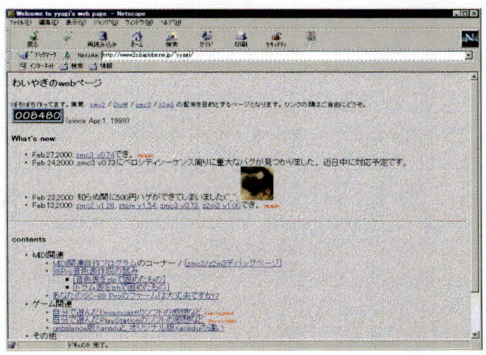
ZMC3/Z2M3/ZEDIT

この快適なZ-MUSIC互換のMMLでのデータ制作環境をWindowsでも実現してくれるのが今回紹介させていただくプログラムです。

まずは，やぎ。氏の開発されたZMC3.EXE。これはMMLで記述されたソースファイル（拡張子.ZMS）をZ-MUSIC形式の演奏ファイル（拡張子.ZMD）にコンパイルするプログラムです。それをさらにWindowsなどで







も標準で扱えるSMF形式（拡張子.MID）にコンバートしてくれるのが，同じくやぎ。氏のZ2M3.EXEです。Windowsのみで扱うならZMDファイルは必要ないので，2つのプログラムはワンセットと考えてよいでしょう。

この2つがあればもうデータ制作はできるのですが，DOSアプリケーションのため，Windowsで使う場合はDOS窓を開いてコマンドを入力したり，バッチファイルを作っていちいち実行する必要があります。

そこで，MASA氏の開発されたZEDIT.EXEを使うと便利です。こちらはテキストエディタにZMC3/Z2M3の呼び出し機能と制作したSMFの演奏機能がついています。これを使えば修正→演奏確認の繰り返しがキーボードから手を離さずに行え，非常に快適です。

Z-MUSICにはVer.2とVer.3があり，MMLレベルでは互換性がありますが，その他の仕様は完全に別物です。ここではVer.3に相当するZMC3.EXE/Z2M3.EXEを中心に説明しますが，付録CD-ROMにはVer.2相当のZMC2.EXE/ZtoM.EXEも収録されていますので必要に応じて使い分けてください。これからMMLに挑戦するという方はZMC3.EXE/Z2M3.EXEのみを使えばよいでしょう。

また，現段階ではX68000とオリジナルのZ-MUSICで作られたデータの再現性には問題がある場合があります。特に過去のOh!X LIVE in掲載作品などで高度なテクニックを使っているものは正しく再生できなかったりエラーになる場合があります。ご注意ください。

★導入

Windows95/98での導入方法です。まずはハードディスクの適当なフォルダに各プログラムと関連ファイルをすべてコピーしてください（ZMC3，Z2M3のみを使う場合でもZMC2，ZtoMもコピーしておいてください）。

次にZEDIT.EXEを起動してください。そしてメニューバーの「オプション」から環境設定を選び，ZMC2.EXE，ZtoM.EXE，ZMC3.EXE，Z2M3.EXE へのパスをそれぞれ指定します。MSヘルプファイルにはdoc¥zmc2man¥zmsman¥index.html をZEDITヘルプファイルにはdoc¥zedit.htm を指定します。ZMSを関連付けしておくとZMSファイルをダブルクリックするだけでZEDITが起動します。

Ver.2とVer.3環境は必要に応じて選択しますが，ここではVer.3環境を前提として説明していきますので，Ver.3のほうを選択しておいてください（下図）。

次にメニューバーの「コンパイル」からZTOM設定を選びます。意味のわかる人は必要に応じてチェックしてください。わからない人はとりあえず以下のように設定してください。「共通コマンド部Exclusiveウエイト」をチェックして6を設定，「ドライバが送る初期化命令も埋め込む」および「数値あり/なし」をチェックして数値は0を指定します。ほかの項目はチェックしません（下図）。

これで準備はできました。あとはエディタ部分にZMSデータを書くかファイルを読み込んで，ツールバーのコンパイル（F5キー），演奏（F1キー）ボタンを押せば自動的にSMFが生成され，演奏が始まります（エラーがなければ，ですが）。

[Column]Windows95/98での注意点

Windows95/98上において，ZEDITの現在のバージョンでは約50KB以上のサイズのファイルは読み込めません。また，ZMSファイルが"Program

X68000でも……

「WindowsでもZMSデータ制作ができるのはわかったが，Windowsでの演奏はテンポが不安定で気に入らない」という方もいるでしょう。私もそのひとりです。レコーディングのときなどはX68000での演奏が望ましいですね。そんなときはZMC3.EXEで制作したZMDファイルをX68000に持っていってZ-MUSIC Ver.3で演奏すればOKです。特殊なコマンドを使わない限り，ZMDレベルでは互換性があります。ただしZ-MUSIC Ver.2（ZMC2/ZtoMが相当）とVer.3（ZMC3/Z2M3が相当）間では互換性はないので注意してください。

また，ZMC3/Z2M3/ZMC2/ZtoMはX68000用の実行ファイルも同梱しています。ZMC3の拡張機能を使ったデータ制作もX68000上で可能です。

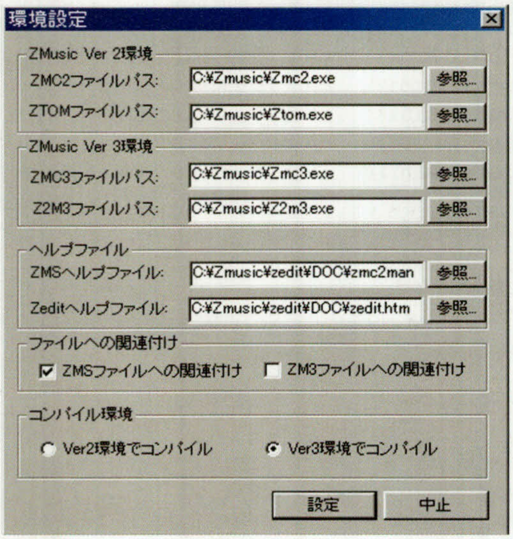
UNIXでも……

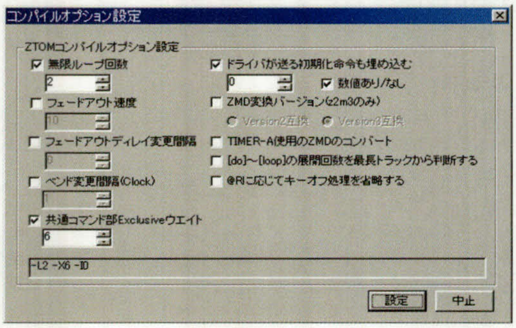
この記事では混乱を避けるためWindows環境を前提に解説していますが，ZMC3/Z2M3はソースが公開されており，UNIX上でもそのままコンパイル可能です。つまり，UNIXでもZ-MUSICによるデータ作成環境が実現できます。

コンパイルが面倒な方には，Debian GNU/Linuxなバイナリファイル（.debファイル）も有志の手により作成されていますので，これを用いるのが便利です。

一方，ZEDITはUNIX上では動作しませんが，代わりにEmacs用のzmusic-mode.elが我孫子氏（http://www.ddna.is.tsukuba.ac.jp/~wabi/Private/）によって作成されており，Emacsのフル機能を利用しながらのデータ作成が可能です。

ちなみに，現在Mac版を作成された方はいらっしゃいません。どなたか挑戦されませんか？





Files", "My Documents" など名前に空白が入っているフォルダ以下にあると正しくコンパイルできない場合があります。WindowsNT4.0ではこの限りではありません。

★データ制作

簡単になりますがZMSデータの書き方を説明します。まずはZMSコマンドとMMLのリファレンスが必要ですので，2000春号のCD-ROMにも収録されているマニュアルドキュメント（Z-MUSIC HOME PAGE（http://www.z-z-z.gr.jp/zmusic/）でZ-MUSIC Ver.3マニュアルアーカイブをダウンロード可能）「MEASURE4 ZMSコマンド（ZM4.MAN）」と「MEASURE5 MMLコマンド（ZM5.MAN）」を参照してください。ただし，一部未対応のコマンドもあります。

また，ZMC3/Z2M3/ZEDITではMIDI音源のみによるデータ作成を想定しているため，X68000のFM音源とAD PCMに関する部分は機能しません。ほかの項目はX68000用なので必要ありません（MEASURE1あたりは参考までに読んでおくといいですね）。

ZMSデータはZMSコマンドによる共通コマンド部とトラックにMMLを書き込むMML部の2つに分かれます。前半にくるZMSコマンド部にはトラックのMIDIチャンネルへのアサインや，音源の初期化や設定のためのエクスクルーシブデータなどを記述します。特に変わったことをしない場合はトラックの定義と音源の初期化メッセージだけでことは足ります。

例）

```
.ASSIGN 1 MIDI1      /トラック1にMIDIチャンネル1を定義
.GM_SYSTEM_ON        /GM初期化
```

音源の初期化のようなよく使われるエクスクルーシブメッセージはこのような専用コマンドも使えます。

それから後半のMML部分。定義されたトラックにMMLを書き込むには .TRACK n{ }コマンドを使い，{ }の中にMMLを記述していきます（従来どおり（Tn）コマンドも使えます）。

例）

```
.TRACK 1 {           /トラック1に書き込む
[PROGRAM 17]         /音色番号17番（オルガン）を選択
[VOLUME 120]         /音量を120に設定
O4                   /オクターブを4に設定
L4                   /デフォルト音長を4分音符に設定
```

リスト/サンプル曲

［リスト1］

```
/ sample list

.comment ZMS sample for GM module by mute

.ASSIGN 1 MIDI1    /トラック1にMIDI Ch.1を定義
.ASSIGN 2 MIDI2    /トラック2にMIDI Ch.2
.ASSIGN 3 MIDI3    /トラック3にMIDI Ch.3
.ASSIGN 4 MIDI10   /トラック4にMIDI Ch.10(ドラムパート)
.ASSIGN 5 MIDI10   /トラック5にMIDI Ch.10(ドラムパート)

.GM_SYSTEM_ON       /GM初期化

/↑共通コマンド部
/-------------------
/↓MML定義部

/----ベース
.TRACK 1 { T146          /テンポ146bpm(1チャンネルのみ指定)
[PROGRAM 39]             /音色39番(Synth Bass 1)
[VOLUME 100]             /音量100
O2 L8                    /オクターブ2, デフォルト8分音符
|:8g:||:8f+:||:8e:||:8d:|  /|:n～:|はn回繰り返し
|:8c:||:8d:||:8g:|g1     /省略時は2回
}

/----メロディ
.TRACK 2 {
[PROGRAM 82]             /音色82番(Sawtooth Lead)
[VOLUME 110]
O4 L8
g4.d4.b4a4.gf+2          /4.は付点4分音符
e4.>b4.<f+gf+4.ed2       /<はオクターブアップ, >はダウン
c>g<ce4cegf+df+a4f+ga
g1                       /1は全音符
}

/----コード(ピアノ)
.TRACK 3 {
[PROGRAM 1]              /音色1番(Grand Piano)
[VOLUME 100]
O4 L8
'dgb'r4'd2gb'r           /'dgb'は和音, rは休符
'df+a'r4'd2f+a'r         /'  '内の音階を同じに発音
'egb'r4'e2gb'r           /音長はどれかひとつに指定
'df+a'r4'd2f+a'r
'eg<c'r4'e2g<c'r
'df+a'r4'd4.f+a''d4f+a'
'd1gb'
}

/----ドラムス
.TRACK 4 {               /GMではドラムは音色選択なし
[VOLUME 127]
O2 L8
|:|:3c4e4cce4:|          /繰り返しはネスト可
| cce4cbgf:|             /繰り返しの最後に | で抜ける
'c4e'
}

/----ハイハット/シンバル
.TRACK 5 {               /音量はトラック4と共通(MIDICh.10)
O2 L8
<c+*0,8>                 /*は絶対音長指定。0の場合持続時間0
|:16f+,,127f+,,100:|     /音符の第2パラメータは発音時間(ゲートタイム)
<c+*0,8>                 /第2パラメータはベロシティ(強さ)
|:12f+,,127f+,,100:|
<c+1
}
```

```
cdefedcr            /演奏部分
efgagfer
crcrcrcr
L8 ccddeefferdrc4   /先頭で音長を8分音符に設定
}
```

このようにトラックごとにMMLを書いていきます。初めの音色，音量は @17 @V120 という表記もできます。というかそれが本来のMMLなんですが，Z-MUSIC（ZMC3）では可読性の高い記述もできるようになっています。

演奏部分のcdefgabはそれぞれ ドレミファソラシ に，rは休符に対応します。このコマンドに続けて数値を書くと何分音符かの指定になります。数値を書かない場合はLコマンドで指定した値になります。

いきなり LIVE in 掲載のリストなどを見ると頭が痛くなると思うので簡単なGM音源用のサンプル（リスト1）をつけておきます。なるべく注釈を入れてますが，各コマンドがなにを意味しているのかマニュアルで調べながら研究してみてください。

これだけでは全然説明不足とは思いますが，最後に紹介している私のホームページではZMSデータ制作に関するコンテンツを公開しています。質問なども受け付けていますのでインターネット接続環境のある方はぜひご訪問ください。

★ZMC3独自の拡張

このほか，ZMC3.EXEではオリジナルのZ-MUSICにはない便利な拡張機能が追加されています。ここではその一部を紹介します。詳しくはZMC3EXT.TXTを読んでください。

・qコマンドによるゲートタイム（クォンタイズレイト?）の先行指定

ベロシティシーケンスと同じく，クォンタイズの先行指定を用いることができます。

たとえばq8,7,5,8,-10,*2 といった指定が可能です。

（q-10 はV3と同じ（10tick短く発音）。Q＊2は@q2と同じ）

・和音のベロシティ指定の拡張

z100,+10,+10 "ceg"

などとすればc,e,gそれぞれに100,110,120のベロシティが割り当てられます（'ceg'だと，c,e,gのベロシティはすべて同じ）。

・ディレイ付き和音の鳴らし方の拡張

和音のディレイに負値を指定すると，構成音を逆順に発音します。たとえば，

'c4eg',-3

と，

'g4ec',3

は同じ音の鳴り方をします。ギターのストロークプレイなんかにどうぞ。

・総利用音長の予約

これではよくわからないかと思いますが，たとえば|| c4e4g4<c4 ||*192などとすると，|| .. ||の中にある音符が全部あわせて全音符分になるかをチェックしてくれるという機能です。

||..||内の総音長（c4e4g4<c4）と予約音長（*192）が一致しない場合はWarningを出力します。

||c4e4g4<c4||1 という表記や，||c4e4g4<c4e4||1^4 という表記も可能です。ネストもできます。和音や連符，ポルタメントなどを中に入れても構いません。

・裏パート演奏処理（トラックの入れ子記述）

任意のmml演奏を裏に回すことができます。これにより，「同じチャンネルの裏でこの音を鳴らしつつ……」といった記述を（見かけ上）1トラックで行えます。たとえば，

[back 'c1eg'] gag<c

とすると，

(t1) w2 gag<c

(t2) w 'c1eg'

相当の出力が得られます。ギター演奏や，キーボードの右手左手のmml記述などに便利です。

このほかXG音源専用コマンドなど新しい機能が加えられています。コマンドやMMLに対する要望やアイデアがありましたらぜひご連絡ください。実現できるとは限りませんが。

※ZMC3の拡張機能を使ったデータを公開するときはZMC3専用であることを明示するよう注意してください。

★最後に

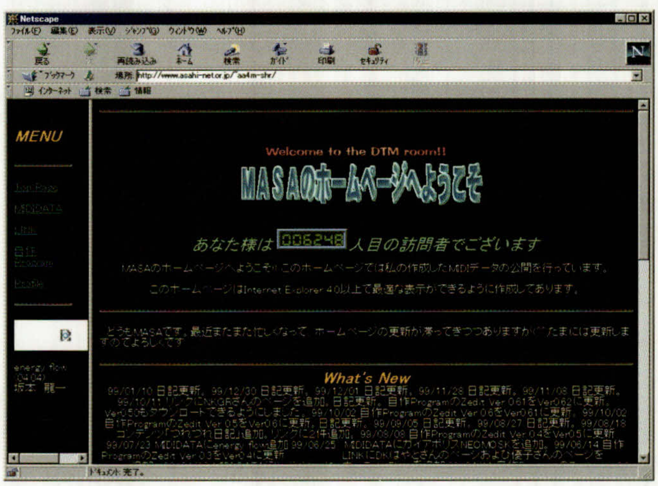
今回紹介したプログラムはまだ開発段階のものです。不具合やオリジナルのZ-MUSICとの相違点などがあるかと思います。疑問点があった場合，インターネット接続環境のある方は作者さんのページで最新版をチェックして，各ページやドキュメントにある「既知の問題点」をよく読んだうえでご連絡ください。

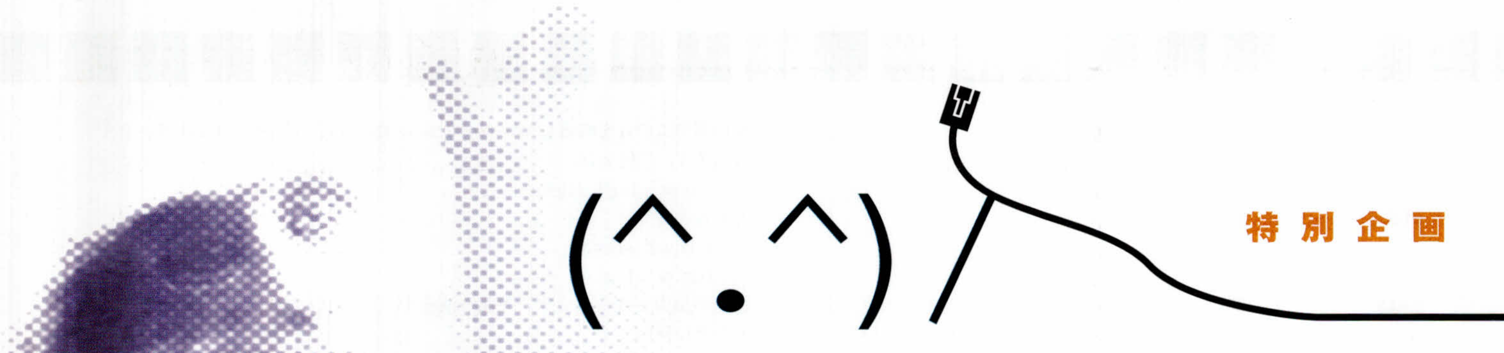
また，拡張機能の提案や要望などの声もお待ちしています。

●ZMC3.EXE/Z2M3.EXE最新版と情報はこちら
わいやぎのWebページ
http://www2s.biglobe.ne.jp/~yyagi/

●ZEDIT.EXE最新版と情報はこちら
MASAのホームページ
http://www.asahi-net.or.jp/~aa4m-shr/

●報告，質問，要望などこちらの掲示板でも受け付けます。
mu-Z studio
http://www02.so-net.ne.jp/~mute/





特別企画

インターネット常時接続への道

猿でも引けるOCNエコノミー

西川 善司 Nishikawa Zenji

情報収集などに便利，といったレベルから，現在ではなにをするにも切り離せなくなってきているインターネット。理想はやはり常時接続。手軽な専用線，OCNエコノミーについて，申し込みから，サーバの構築まで見てみましょう。

検討編

インターネット使ってます?

「インターネット」という言葉を10秒に1回は聞く昨今，いかがお過ごしだろうか。

私，西川善司はもはやインターネットのない生活など考えられない状況になっており，朝起きたら顔洗う前にインターネット，食事の前のオードブルにインターネット，食事中もインターネット，食後のデザートもインターネット，仕事の前にインターネット，仕事中もインターネット，仕事の息抜きにインターネット……という具合に，インターネットしていない時間はウンコしているときと眠っているときくらいのもの。ところが，眠っているときにはインターネットの夢を見ているし，脱糞時の爽快感さえもインターネットで世界に情報発信して伝えたい……と考えるほどのインターネット中毒に陥っている。

さて，皆さんはインターネットに接続するときにどういう方法を使っているだろうか。

もっとも多いパターンは，ISPと呼ばれるインターネット接続サービス業者と契約し，電話回線を通じてアクセスする方法だろう。

会社勤めの人や学生は，会社や学校のLANからインターネットにアクセスしているかもしれない。こういう人でも，自分個人のインターネット接続環境は電話回線を使ったISP経由の接続形態の人が多いと思う。

このほか，ケーブルテレビ業者がケーブルテレビ回線によるインターネット接続サービスを提供している場合もあるが，利用できる地域がまだ限られており，普及しているとはいいがたい。結局，電話を使ってのISP経由接続というのがもっとも一般的なスタイルといえる。

ところが，1996年12月にひとつの新しいインターネット接続スタイルがNTTより提供された。それが「OCNエコノミー」だ。

OCNエコノミーってなに?

OCNエコノミーは，いわゆるインターネットの専用線接続サービスの一種だ。専用線なので，その線は当然のごとくインターネットに常時接続される。プロバイダはいらないし，電話代もかからない。

ただ，「専用線」というのは「利用コストが高い」というイメージがある。ところがOCNエコノミーというサービスは，これを低価格で実現してしまったのだ。

OCNエコノミーの通信速度は128kbps。これはNTTが提供するISDN回線サービス「INS64」の2倍の通信速度に相当する。ちょうどISDN回線チャネルを2つ使った通信速度であるMP（Multilink PPP）方式の伝送速度とほぼ同じというわけだ（なお，このほか，OCNエコノミーよりも上位であるOCNスタンダードは1.5Mbps，OCNエ

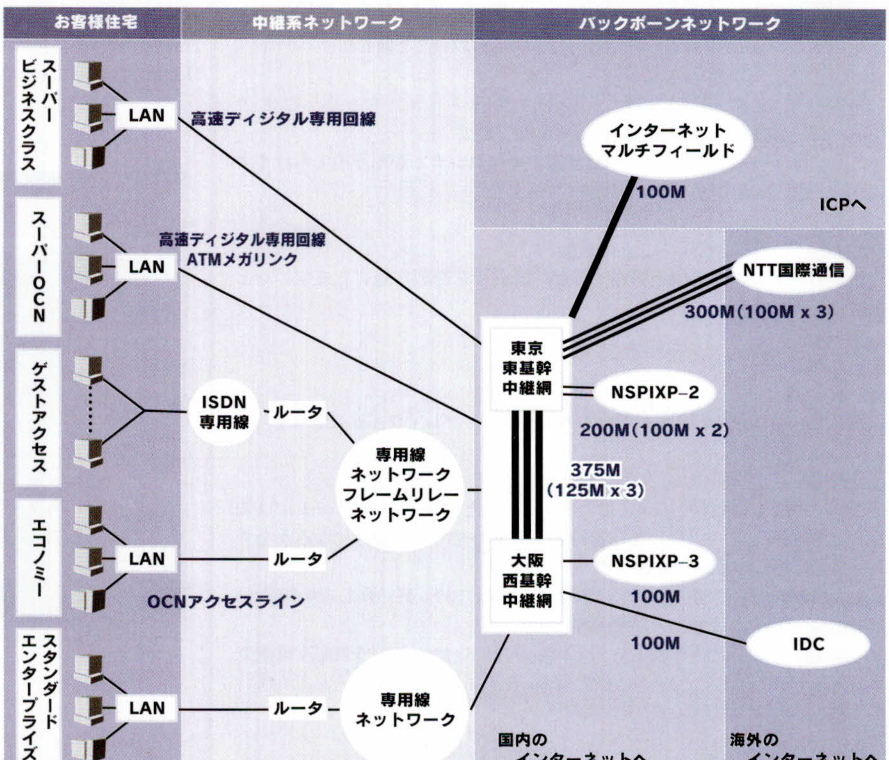


図1

ンタープライズは6Mbpsの通信速度で提供されている)。

インターネット専用線というと数Mbpsの通信速度を提供するものが多かったわけだが，OCNエコノミーはこの通信速度を128kbpsに抑える代わりに値段も安くした……ということだ。

ただし，ひとつ注意したいのはOCNは通信速度を保証しない「ベストエフォート(Best Effort)型」の通信サービスであるという点。「ギャランティ(Guarantee)型」とは違い，「自分以外のアクセス者の回線使用状況によっては128kbps以下になることもあるよ」というサービスなのだ。

図1を見てもらうとわかるがOCNエコノミーの加入者からの線はNTT側のルータに接続されているが，このルータに最大で24ユーザーがアクセスする形になっているのである。つまり，ある許容量を持った通信線を最大24人で使うことになるわけで，誰も使っていないときは128kbpsの速度が出るが，利用者が多いときは128kbps以下になることもある……ということなのだ。

実際にはかなりのマージンを持って運営しているようで，NTT側のルータに24ユーザーを接続させている例は少ないようだ。ただ，NTT側のルータとその上位のフレームリレーとの接続速度も128kbpsに抑え込まれているため，どんなに回線が空いていても128kbpsを超える通信速度が得られることはなく，さらに回線が混雑した場合はその速度の落ち込みが急激になることを予想させる。

なお，筆者の場合，OCNエコノミーでサイトを構築して2年が過ぎており，毎日の利用頻度は相当なものにも関わらず，極度の通信速度の落ち込みというものは，いまのところ経験していない。

まぁとにかく，(やや誤解を含んだいい方になるが)「OCNエコノミーはISDNの2倍の通信速度のインターネット専用線」ということになる(図1)。

OCNエコノミーって本当に安いの?

「低価格ってどのくらい?」という質問が出るのはごもっとも。料金体系はリスト1のようになっている。NTT以外の日本テレコム，DDIといった電話会社も，OCNアクセスラインを使った同種サービスの提供を開始しているので，こちらの料金体系も同じ表にまとめておく。

さて，NTTのOCNエコノミーでは初期費用は別として月々32,000円(税別)がかかるということになる。

「低価格といってもそんなに安くないじゃん」という意見も出てくるだろうが，それはもっともな意見だ。

では，ISDN回線を使ったインターネット接続環境とOCNエコノミーについて，簡単なコスト比較を行ってみよう。

まずはISDN回線のほうのコスト計算から。

月々のISPの基本利用料金が5000円で使い放題とする(例：SO-NET)。ISDN回線はINS64の場合，月々の基本回線使用料が2830円だ。プロバイダのアクセスポイントに対しINSテレチョイスを契約していたとすると，これの基本料金が200円で，通信費は30%割引されることになる。通信費は3分10円。1日4時間使ったとすると1カ月で120時間，この通信費は24000円だ。このときのインターネット接続にかかったコストは，

$$5000+2830+200+24000\times0.7=24840$$

となる。1日の接続時間が6時間の場合は，

$$5000+2830+200+36000\times0.7=33240$$

となる。

OCNエコノミーのは月々32,000の固定料金なので，この概算結果から考察すると，だいたい，1日に6時間以上インターネットを接続する利用環境であれば「OCNの導入は現実的だ」ということになる。

はいはい。

「低価格，低価格と繰り返しておいて電話回線を使ったISP経由接続のほうが全然安いじゃんかーコラー」といきり立つ人が出てくるのもごもっとも。テレホーダイ系のサービス時間だけ使う人ならさらに格段に安くできる。実際，インターネットを1日に1時間以下しか使わないカジュアルユーザーならば絶対に一般的なISP経由接続のほうが安くて現実的なのだ。

ただ，専用線ならではの魅力があることも忘れてはならない。

専用線の魅力とは

OCNエコノミーは専用線だ。何度もいうようだが。この専用線の魅力について語ってみよう。

●365日，24時間繋ぎっぱなし

OCNエコノミーの魅力は第一にずっと繋ぎっぱなしでいいという点にある。

数十MBのファイルサイズのデモプログラムやフリーソフトなどが平気でリリースされることは珍しくはないが，実際にこれを電話回線を使ったISP経由の接続方式でダウンロードするのには無理がある。これが専用線であれば，繋ぎっぱなしでOKで，そのまま外出してしまうことだってできる。

●自分のパソコンをWWWサーバやメールサーバにできる

そしてユーザー側でサーバを持てるというところにも大きな魅力がある。

OCNエコノミーでは，契約者が不変のIPアドレスを取得できるため，ユーザー側のパソコンをWWWサーバやメールサーバとしてインターネット上に公開できるのだ。これまた誤解覚悟でいってしまうと「ユーザー側がミニプロバイダ(ISP)になってしまう」みたいな感じだ。

自分のパソコンをWWWサーバとして公開できるということは，作れるWebページの容量制限がなくなることを意味している(サーバとして公開する，そのパソコンのHDD容量に制限されるが)。

また，ISP経由接続ではセキュリティ上の問題で制限されていた，掲示板をはじめとしたCGIプログラムや，その他の高度なインターネット向けインタラクティブコンテンツだって公開できる。

ISP経由接続の場合だと禁じられていることの多い商用ホームページなんかもOCNエコノミーの場合ならばもちろんOKだ。

そしてメールサーバを自分で持てるということ

リスト1　OCNアクセスラインを使った低価格専用線サービス

会社名	NTT	DDI	日本テレコム	KDD
サービス名	OCNエコノミー	DIONスタンダード	ODNエコノミー	NEWEBアクセス3
サポートWeb	http://www.ocn.ne.jp/	http://www.dion.ne.jp/	http://www.odn.ne.jp/	http://www.neweb.ne.jp/
回線速度	128kbps	128kbps	128kbps	128kbps
IPアドレス取得可能数	8個または16個	基本的には8個	基本的には8個	最大16個
初期費用(条件によって異なってくるため，あくまで一例とする)	12600円(契約料，基本工事，屋内配線，機器工事の合計)	16600円(回線工事費，契約料，基本工事費，交換機工事費，屋内配線工事費の合計)	12200円(工事費，基本工事，屋内配線工事，交換機工事，契約料の合計)	3000円(基本工事費)
月額費用	32000円	30900円	30900円	30900円
ドメインネーム取得代行費用総額	6000円	6000円	6000円	6000円
IPアドレス取得代行費用総額	6000円	6000円	6000円	8000円
セカンダリDNSサーバ設置料	0円	0円	0円	1000円
ドメイン名維持管理手数料(年額)	5000円/1ドメイン名	0円	5000円/1ドメイン名	6000円(1ドメイン名/月額500円)

は，メールのアカウントも自由に作れるということだ。もちろんメーリングリストの主催も思いのまま。

ISP経由接続のメールサービスでは送受信できるメールの容量に制限があったが，自分のパソコンをメールサーバにできるので，これもない。たとえば数十MBのアーカイブファイルを添付したメールを送受信することだって可能なのだ。

●家族それぞれのパソコンがインターネット常時接続になる

OCNエコノミーへユーザー側で設置したルータを介して接続しているのであれば，最大通信速度の128kbpsの範囲で，ほかのパソコンやそのプライベートLANのメンバーが同時にインターネットにアクセスすることもできる。

上でもちょこっと触れたが，OCNエコノミーを契約すると最大16個のIPアドレスを取得できるので[*1]，これを家族のパソコンに割り当てれば，これらもインターネットに常時接続できることになるのだ。たとえば兄弟2人で割り勘したとすれば月々1人16000円，3つ子の兄弟ならば1人10666円，父母+2人兄弟の4人で割り勘したとすれば1人8000円だ。これならば常時接続もグンと現実的な選択になる。

(*1) 2つはブロードキャスト用にリザーブされ，ひとつはルータ(ゲートウェイ)にリザーブされるので，クライアントPCに割り当てられるのは13個ということになる。

●独自ドメイン名が持てる

ドメイン名とはsoftbank.co.jpみたいなインターネット上の固有のネットワーク名のこと。OCNエコノミーに契約すればこれがひとつ取得できるのだ。

いままでは自分のホームページは，

http://www.zo-net.ne.jp/~zenji/

のようなISPの名前の下階層になっていたのが，

http://www.zenji.gr.jp

のようにできるということだ。ドメイン名はJPNICという組織が一括管理しており，すでに取得されてしまっているものはダメだが，とにかく自分で考えた名前を自分のネットワークにつけられるというのは大きな魅力だといえる。

たとえばインターネット上でのビジネスを考えているならば，社名にしたり，自分たちのチーム名や組織名にしたりすればいい。そしてその名刺には，自分たちで考えたドメイン名つきのホームページアドレスやメールアドレスが記載できることになり，これはちょっとカッコいい(やや自己満足気味のカッコよさだけどね)。

契　約　編

契約の前に

低価格専用線を引くとして，本当にOCNエコノミーでいいかをいま一度検討してみよう。

197ページのリスト1に示したようにOCNエコノミーには同種の競合サービスがある。

日本テレコムは「ODNエコノミー」，DDIは「DIONスタンダード」[*2]，KDDは「NEWEBアクセス3」という名前でサービスが提供されている。こうした競合サービスはNTTのOCNエコノミーよりも価格が10%前後安めに設定されているのが特徴だ。OCNエコノミー自身もこれら競合サービスに対抗するために，3年間の長期契約をすれば，月々の料金が2310円(約8%)も安くなるという「長期継続利用割引サービス」(http://www.ocn.ne.jp/ocnweb/news/contents/keizoku.html)を1999年11月から開始している。こうしたサービスを利用すれば他社の競合サービスよりも安くなることも覚えておこう。

また，NTT以外のOCNエコノミー競合サービスはサポートされている地域がだいぶ限定されてしまっている点にも注意したい。契約前に自分が住んでいるところがサービス対象地区か確認する必要があるのだ。

ODNはサービスインフォメーションのWebページ(http://www.odn.ne.jp/infoodn/leased1.html)で，そしてNEWEBアクセス3はSERVICE INFOのページ(http://www.neweb.ne.jp/corporate/access/ap03.html)で，サポートしている区域の電話番号の市外局番と市内局番の表を公開している。DIONはユーザーの電話番号からサポート地区かどうかを判定するWebページ(http://w11.dion.ne.jp/lan_service/service/standard/densrc.html)を公開しているのでこれらを参照するといい。

(*2) なお，DIONは「エコノミー」でなく「スタンダード」がOCNエコノミーに相当する点に注意(まぎらわしい)。

OCNエコノミーを申し込む

OCNエコノミーの導入を決定したら，早速契約をしよう。

まずは申込書を手に入れる必要がある。これはOCNインフォメーションデスク(フリーダイアル：0120-047816)へ電話をかけ，「OCNエコノミーの資料と申し込み用紙を送ってください」といえば郵便で送ってもらえる。

そしてOCNエコノミーを導入するためには「ISDNルータ」という装置が必要不可欠になる。これもPCショップや秋葉原などの電気街で事前に購入しておく必要がある。機種にもよるがDSU内蔵タイプならば実売価格で3万円から4万円の間くらいだ。現在そのコストパフォーマンスの高さから人気があるのは，

- NTT-ME MN128-SOHO SL11(http://www.ntt-me.co.jp/mn128/slev_index.html)定価49,800円(実勢販売価格32,800円)
- YAMAHA RTA50i(http://netvolante.rtpro.yamaha.co.jp/)定価49,800円(実勢販売価格37,800円)

の2機種。人気機種だけに取扱店やWeb上の関連情報も多いので，どちらかを選んでおけば問題はないはずだ。

お得なISDNルータ+OCN申し込みパック商品

「どれを買っていいかわかんないし，面倒くさい」という人におすすめなのが，パソコンショップ「TWOTOP」が行っているOCNエコノミー仲介パック(http://www.twotop.co.jp/special/ocn.asp?mscsid=0VERS9D3RWS12K4C00J74EEEM15CAAB3)だ(図2)。

このOCNエコノミー仲介パックは，その名前のまんま，OCNエコノミーの申し込み仲介サービスとOCN対応のISDNルータをひとつにまとめた商品だ。選択できるISDNルータは前述のMN128-SOHO SL11かYAMAHA RTA50iの2機種どちらかで，仲介料込みのパック料金はなんと26,600円(税別)だ。

ISDNルータ単体の定価が49,800円なのに，このパック価格だと仲介手続き込みで26,600円になってしまうという不思議なセットなのである[*3]。ISDNルータは単体で秋葉原などの電気街で買っ

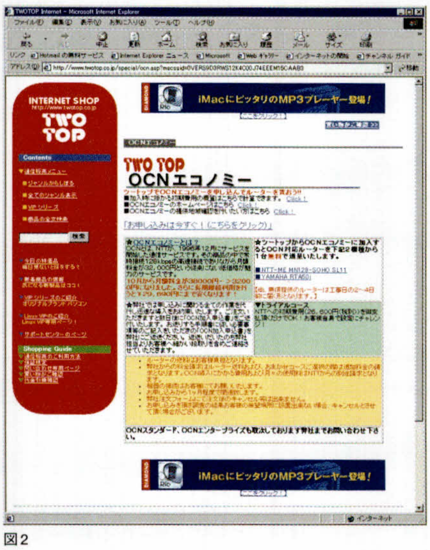


図2

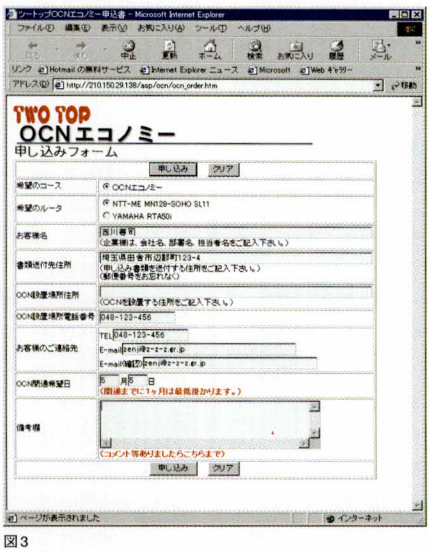


図3

てもこの価格より高いくらいなので，これは絶対的にお得なパックだといえる。

この仲介パックは基本的にオンライン販売の形態をとっているので，インターネットから申し込む形になる。図3は，このパックの申し込み用フォームのページに実際にサンプルとして必要事項を記入してみたものだ。

(*3)詳しい事情はわからないけど，まぁNTTがいくらか負担しているんでしょうなぁ……。毎月コンスタントに3万円払ってくれる顧客を獲得できるんだから，そりゃあそのくらいはするってもんでしょうなぁ。ちなみにTWOTOPの仲介パックではODNやDIONを契約することはできない。

無料コンサルティングを受けよう

TWOTOP経由で申し込んだ場合，NTTに直接申し込んだ場合，いずれの場合も数日内にNTTから営業の人間が申込書を持ってやってくるはずだ。

このときやってくる人間はOCNエコノミーに関して基本的な知識はだいたい身につけているので（例外はある），わからないことがあればどんどん質問してOCNの謎を明らかにしよう。

私がOCNエコノミーを引いたときの話をちょっと。うちのケースでは，住んでいる場所が埼玉県のイナカで，OCNエコノミーすらそれほど導入例がないらしく，やってきた人間はそれほどOCNに詳しい感じではなかった（いきなり「例外君」にヒットしたわけだ）。よってその場でこちら側の質問には一切答えられなかったのだが，しかし例外君はその「疑問リスト」を持ち帰り，数日後に質問に対するすべての答えを記載した書類をファックスしてくれた。まあ相手が詳しい人にしろ例外君にしろ，いずれにせよ質問には答えてもらえるだろう。ただ，相手がそういう例外君の場合，こちらの質問の意味すら理解できず夢うつつで去っていく危険性があるので，聞きたいことはリストアップし書面にまとめておいたほうがいい。

特に事前に確認しておきたいのが，工事費についてだ。すでにISDN回線対応のメタル線が屋内に配線がしてあり，これが屋外の引き込み口と繋がっている場合，屋内配線工事は不要になる。またDSU内蔵ルータを使う場合は機器工事費が不要な場合もある。初期工事費の項目は非常に定義が曖昧でわかりにくい部分が多いので，1項目ずつその作業内容を確認したほうがいいだろう。

申込書を書こう

それでは申込書に記入していくことにしよう。ここでは，

・プライマリDNSサーバを自宅に置く（ただしセカンダリDNSサーバはOCN側に設置）
・メールサーバ，WWWサーバなどすべてのサーバを自宅に置く
・独自ドメインを取得する
・IPアドレスを16個取得する

という方針で進めていくとする。

申込パターンとしてCASE1〜3が用意されているが（図4），今回の例ではCASE1に当該することになる（記載例として示されている住所や組織名などは実在しません）。

●申込用紙1（お客様情報）

「注文の種類」は「OCNエコノミー（DNSお客様設置）」を選ぶこと。

この用紙でもうひとつ注意したいのは「アクセスポイント」だ。これはOCNのWebの「サービス提供地域」のページ（http://www.ocn.ne.jp/ocnweb/service/eco/numbers1.html）にて，事前に調べておく必要がある。ここに自分の家の電話番号区域が掲載されていない場合は，NTTに別途確認する必要がある。ここに記載されていないからといって諦めないこと。ちなみに私が回線を契約したときは，このリストすらない状況だったが無事開設できた。

●申込用紙2（お客様DNSサーバ設定情報）

「使用するドメイン名について」では，独自ドメインをどういう手段で取得するかを選択する。普通は「1.新たに申請する」のNTT代行申請を利用すればいい。

「使用ドメイン名」「DNSサーバ名」「他DNSサーバ名」「複数ドメイン名接続情報」は新規に取得する場合は記入する必要はない。

「セカンダリDNS」については「希望する」を選択する。プライマリDNSサーバを自分で設置する場合でもセカンダリDNSサーバはOCN側で設置してもらうのが普通だ。

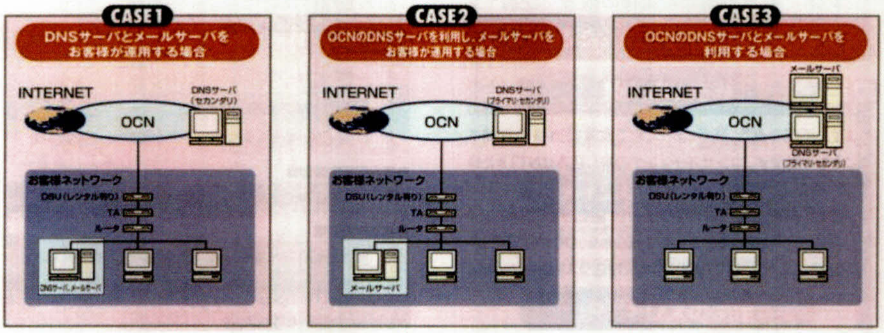
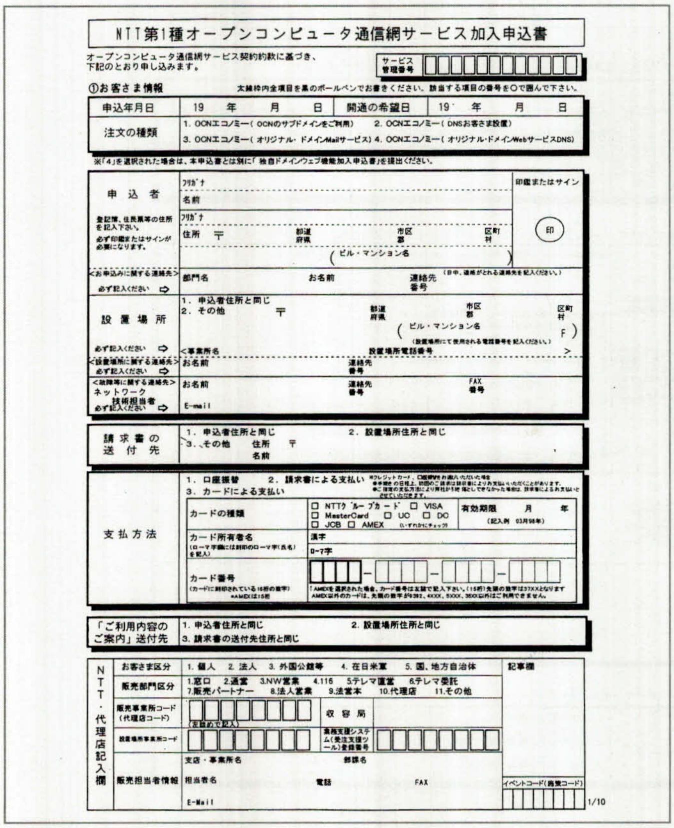


図4



NTT第1種オープンコンピュータ通信網サービス加入申込書

オープンコンピュータ通信網サービス契約約款に基づき、下記のとおり申し込みます。　サービス管理番号

①お客さま情報　太線枠内全項目を黒のボールペンでお書きください。該当する項目の番号を○で囲んで下さい。

申込年月日	19　年　月　日	開通の希望日	19　年　月　日
注文の種類	1. OCNエコノミー（OCNのサブドメインをご利用）　2. OCNエコノミー（DNSお客さま設置） 3. OCNエコノミー（オリジナル・ドメインMailサービス）4. OCNエコノミー（オリジナル・ドメインWebサービスDNS）		

※「4」を選択された場合は、本申込書とは別に「独自ドメインウェブ機能加入申込書」を提出ください。

申込者 登記簿、住民票等の住所を記入下さい。 必ず印鑑またはサインが必要になります。	フリガナ 名前 フリガナ 住所 〒　都道府県　市区郡　区町村 （ビル・マンション名）	印鑑またはサイン 印
<お申込みに関する連絡先> 必ず記入ください ⇨	部門名　お名前　連絡先番号（日中、連絡がとれる連絡先を記入ください。）	
設置場所 必ず記入ください ⇨	1. 申込者住所と同じ 2. その他 〒　都道府県　市区郡　区町村 （ビル・マンション名　F） <事業所名　設置場所電話番号（設置場所にて使用される電話番号を記入ください。）>	
<設置場所に関する連絡先> 必ず記入ください ⇨	お名前　連絡先番号	
<故障等に関する連絡先> ネットワーク技術担当者 必ず記入ください ⇨	お名前　連絡先番号　FAX番号 E-mail	

請求書の送付先	1. 申込者住所と同じ　2. 設置場所住所と同じ 3. その他　住所 〒 名前

支払方法	1. 口座振替　2. 請求書による支払い　3. カードによる支払い		
	カードの種類	□ NTTグループカード □ VISA □ MasterCard □ UC □ DC □ JCB □ AMEX（いずれかにチェック）	有効期限　月　年（記入例 03月98年）
	カード所有者名（ローマ字欄には刻印のローマ字（氏名）を記入）	漢字 ローマ字	
	カード番号（カードに刻印されている16桁の数字）＊AMEXは15桁	□□□□-□□□□-□□□□-□□□□ 「AMEXを選択された場合、カード番号は左詰で記入下さい。（15桁）先頭の数字は37XXとなります AMEX以外のカードは、先頭の数字が9392、4XXX、5XXX、35XX以外はご利用できません。	

「ご利用内容のご案内」送付先	1. 申込者住所と同じ　2. 設置場所住所と同じ 3. 請求書の送付先住所と同じ

NTT・代理店記入欄	お客さま区分	1. 個人　2. 法人　3. 外国公館等　4. 在日米軍　5. 国、地方自治体	記事欄
	販売部門区分	1.窓口　2.通営　3.NW営業　4.116　5.テレマ直営　6.テレマ委託　7.販売パートナー　8.法人営業　9.法営本　10.代理店　11.その他	
	販売事業所コード（代理店コード）	（左詰めで記入）　収容局	
	設置場所事業所コード	業務支援システム（受注支援ツール）登録番号	
	販売担当者情報	支店・事業所名　部課名 担当者名　電話　FAX E-Mail	イベントコード（施策コード）

1/10

図5

●申込用紙3(IPアドレス割り当て申請1/2)

「ネットワーク名」はドメイン名と同じにしたほうがわかりやすい。ただし，この名前が必ずしもドメイン名として取得できるわけではなく，仮にドメイン名と違ってしまっても特に問題はない。

「組織名」は法人の場合は会社名を記入する。法人格を持っていない場合，たとえば個人，クラブ，同好会，法人登録していない自営業などの場合もその団体名を記載しておく。たとえば個人の場合，「おれたち仲よし4人家族」「SF映画研究会」とかそんなのでもいいのだ。

「組織種別」の欄には，その組織が法人格を持っている場合，その会社種別を記載すればいい。個人やそれに準ずる組織の場合は「任意団体」という種別になる。この場合は，

組織名(代表者名)

というフォーマットで記載する。

このあと英語表記でも同様の内容を記載するのだが，注意したいのは住所のところ。英語表記では，

番地　町名　市名　県名　郵便番号　国名

の順序になる。

「組織種別」の英語表記はたとえばこんな感じになる。

株式会社→CORPORATION
有限会社→COMPANY(LIMITEDをつけても可)
任意団体→GROUP

「ネットワークプラン」はだいたいを適当に書いておけばいい。IPアドレスを16個取得する場合は，ホスト(ネットワークに参加するパソコン)の台数が1年以内に10台を超えるような見積もりを書いておきその必要性をアピールしたほうがいい。

「ネットワーク構成図」の図も適当でいい。新規に専用線を引いた場合は「お客さまネットワークのインターネット接続について」のところは「1.まったくの新規接続」を選択しておく。

このあと「IPアドレス割り当て申請1/2)」という申込用紙があるが，これはなんらかの手段ですでに取得しているIPアドレスを今回加入するOCNエコノミーのネットワークとリンクさせる場合の情報を書き込むものだ。今回の例では関係ないので解説は省略する。

●申込用紙4(ドメイン名取得申請書)

これは新規にドメイン名を取得する場合に絶対に記入しなければならない書類だ。

「希望ドメイン名」は，自分たちのネットワークがどういう名前でインターネット上へ公開するかを決めるものだ。他人に取得されているものは使えないので，事前にある程度調査をしておくこと。

なお，ドメイン名は，

固有名.組織種別名

で表される。

固有名は適当に考えればいい。ただし，ありそうな名前はすでに取得されている場合がほとんどだ。

組織種別名は法人であれば「CO」，任意団体であれば「GR」を選ぶことになる。「CO」ドメインを名乗るためには法人格を取得している必要がある。会社組織であっても法人格を取得していない場合は「GR」ドメインになるので注意が必要だ。

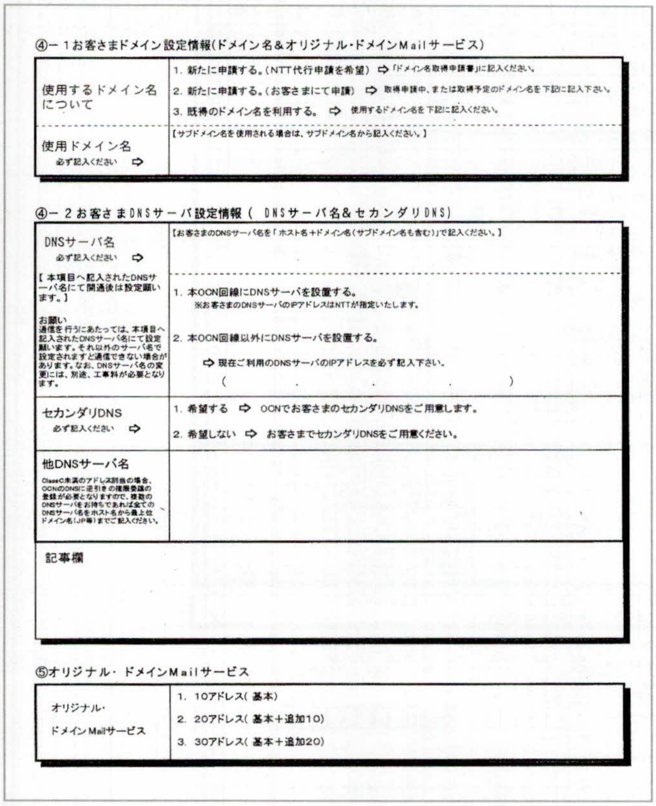
ところで同じ固有名であっても組織種別名が違えばドメイン名としては別名として認識されることを覚えておこう。たとえば「abc.co.jp」と「abc.ne.jp」は違うドメイン名として扱われるのだ。

自分の考えたドメイン名がすでに取得されてしまっているかどうかを確認するには図9の「JPNIC Whois Gateway」(http://www.nic.ad.jp/cgi-bin/whois_gate)を利用するといい。

使い方は単純で，keyワードに「softbank.co.jp」という感じでドメイン名を入力し，[query]ボタンを押すだけだ。もし，そのドメイン名が取得済みであれば，その組織団体の情報が表示される仕組みになっている。もし，「存在しない」というような検索結果が出れば，そのドメイン名は自分で取得が可能だということになる。

「登記情報」は法人格を持っている団体のみが記入すべき項目だ。「GR」ドメインを取得する個人や任意団体の場合は「代表者情報」「副代表者情報」を記載する。このなかの「登記年月日」はもちろん記入不要だが，「登記地住所」は代表者の住所を書いておく。

なお，もう1枚，「個人情報」を記載する関連申込書があるが，これはただ名前と住所，そのほかの連絡先事項を記入するだけの単純なものなので解説は省略する。



④－1お客さまドメイン設定情報(ドメイン名&オリジナル・ドメインMailサービス)

使用するドメイン名について	1. 新たに申請する。(NTT代行申請を希望) ⇨「ドメイン名取得申請書」に記入ください。 2. 新たに申請する。(お客さまにて申請) ⇨ 取得申請中、または取得予定のドメイン名を下記に記入下さい。 3. 既得のドメイン名を利用する。 ⇨ 使用するドメイン名を下記に記入ください。
使用ドメイン名 必ず記入ください ⇨	【サブドメイン名を使用される場合は、サブドメイン名から記入ください。】

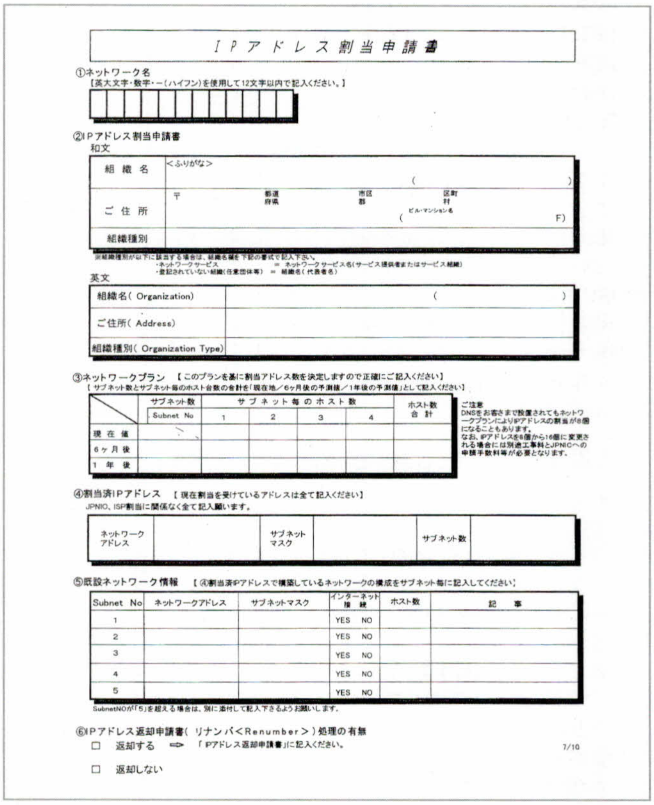
④－2お客さまDNSサーバ設定情報(DNSサーバ名&セカンダリDNS)

DNSサーバ名 必ず記入ください ⇨	【お客さまのDNSサーバ名を「ホスト名+ドメイン名(サブドメイン名も含む)」で記入ください。】
【本項目へ記入されたDNSサーバ名にて開通後は設定願います。】 お願い 通信を行うにあたっては、本項目へ記入されたDNSサーバ名にて設定願います。それ以外のサーバ名で設定されますと通信できない場合があります。なお、DNSサーバ名の変更には、別途、工事料が必要となります。	1. 本OCN回線にDNSサーバを設置する。 ※お客さまのDNSサーバのIPアドレスはNTTが指定いたします。 2. 本OCN回線以外にDNSサーバを設置する。 ⇨ 現在ご利用のDNSサーバのIPアドレスを必ず記入下さい。 (　　.　　.　　.　　)
セカンダリDNS 必ず記入ください ⇨	1. 希望する ⇨ OCNでお客さまのセカンダリDNSをご用意します。 2. 希望しない ⇨ お客さまでセカンダリDNSをご用意ください。
他DNSサーバ名 ClassC未満のアドレス割当の場合、OCNのDNSに逆引きの権限委譲の登録が必要となりますので、複数のDNSサーバをお持ちであれば全てのDNSサーバ名をホスト名から最上位ドメイン名(JP等)までご記入ください。	
記事欄	

⑤オリジナル・ドメインMailサービス

オリジナル・ドメインMailサービス	1. 10アドレス(基本) 2. 20アドレス(基本+追加10) 3. 30アドレス(基本+追加20)

図6



IPアドレス割当申請書

①ネットワーク名
【英大文字・数字・－(ハイフン)を使用して12文字以内で記入ください。】

②IPアドレス割当申請書
和文

組織名	<ふりがな>　(　　)
ご住所	〒　都道府県　市区郡　区町村　ビル・マンション名(　　F)
組織種別	

※組織種別が以下に該当する場合は、組織名欄を下記の書式で記入下さい。
・ネットワークサービス ＝ ネットワークサービス名(サービス提供者またはサービス組織)
・登記されていない組織(任意団体等) ＝ 組織名(代表者名)

英文

組織名(Organization)	(　　)
ご住所(Address)	
組織種別(Organization Type)	

③ネットワークプラン【このプランを基に割当アドレス数を決定しますので正確にご記入ください】
【サブネット数とサブネット毎のホスト台数の合計を「現在地／6ヶ月後の予測値／1年後の予測値」として記入ください】

	サブネット数	サブネット毎のホスト数				ホスト数合計
	Subnet No	1	2	3	4	
現在値						
6ヶ月後						
1年後						

ご注意
DNSをお客さまで設置されてもネットワークプランによりIPアドレスの割当が8個になることもあります。
なお、IPアドレスを8個から16個に変更される場合には別途工事料とJPNICへの申請手数料等が必要となります。

④割当済IPアドレス【現在割当を受けているアドレスは全て記入ください】
JPNIC、ISP割当に関係なく全て記入願います。

ネットワークアドレス		サブネットマスク		サブネット数	

⑤既設ネットワーク情報【④割当済IPアドレスで構築しているネットワークの構成をサブネット毎に記入してください】

Subnet No	ネットワークアドレス	サブネットマスク	インターネット接続	ホスト数	記事
1			YES NO		
2			YES NO		
3			YES NO		
4			YES NO		
5			YES NO		

SubnetNOが「5」を超える場合は、別に添付して記入下さるようお願いします。

⑥IPアドレス返却申請書(リナンバ<Renumber>)処理の有無
□ 返却する ⇨「IPアドレス返却申請書」に記入ください。
□ 返却しない

7/10

図7

サイト構築編〜ハードウェア編

どういうハードウェア構成でいくか

申込書をNTTへ郵送し，申請内容に問題がなければ，しばらくしてNTT側から工事担当の技術者が派遣されてくる。

工事が完了するとNTT側から1通の封書が郵送されてくる。この中には，取得した自分のネットワークの情報が書かれた書類が入っている。

今度は，ここに書かれた情報を使って，いよいよ自分のサイトを構築していくことになる。

●何台のパソコンをサーバにする?

まず，第一に何台のサーバを動かすか決めよう。

今回の例では自宅にDNS，WWW，メールの3つのサーバを設置することにしているが，そのサーバにするパソコンは3台なくてはダメなのだろうか。もちろん3台のパソコンをそれぞれ独立したサーバとして動かすほうが体裁はいいが，別に1台のパソコンに3つのサーバサービスを任せてしまってもいい。

ただし，1台のパソコンで3つのサーバ機能を任せた場合，万が一このパソコンが故障してしまうと，そのサイトは完全に死亡してしまう。3つのサービスを1台ずつ別々のパソコンのサーバで動かした場合ならば，たとえばWWWサーバが故障してしまったとしても，別のパソコンで動いているメールサーバにはまったく影響がないことになるわけで，サイト全体として見た場合，被害は少ない。

サイトの堅牢性をアピールするには3台のパソコンにしたほうがいいだろうが，この場合，OCNエコノミーの契約時に取得した数限りある8個または16個のIPアドレスを3つも消費してしまう。まぁ，さまざまな意見があるだろうがOCNエコノミー程度の小規模なサイトならば1台のサーバでも問題はないはずだ。

今回ここで示す例ではDNS，WWW，メールの3つのサーバ機能を1台のPCで任せる例を示していくことにする。

●どんなパソコンをサーバにする?

サーバパソコンというとすごく値段が高そうに思えるが，OCNエコノミー(128kbps)程度のサーバにするならばそんな高性能パソコンはいらない。Pentium100M〜166MHzもあれば十分だ。むしろ安定性のほうが要求される。なにしろ365日24時間，ぶっ続けで動作させるわけなので，いくら高速でも，ちょっと動かしてハングアップしてしまうようなマシンでは困るのだ。

おすすめなのは，1，2世代くらい前のメーカー製パソコン。電気街で叩き売りされている新古品パソコンなどが狙い目だ。

新品を買わずに，友人などから安く譲ってもらったり，個人売買で手に入れたりするのもいいだろう(故障品はダメよん)。

ただし，ノートパソコンは特殊な事情がない限り避けること。理由は2つ。

ひとつは，ノートパソコンが比較的特殊なハードウェアで構成されていることからサーバOSがインストールできない可能性があるためだ。

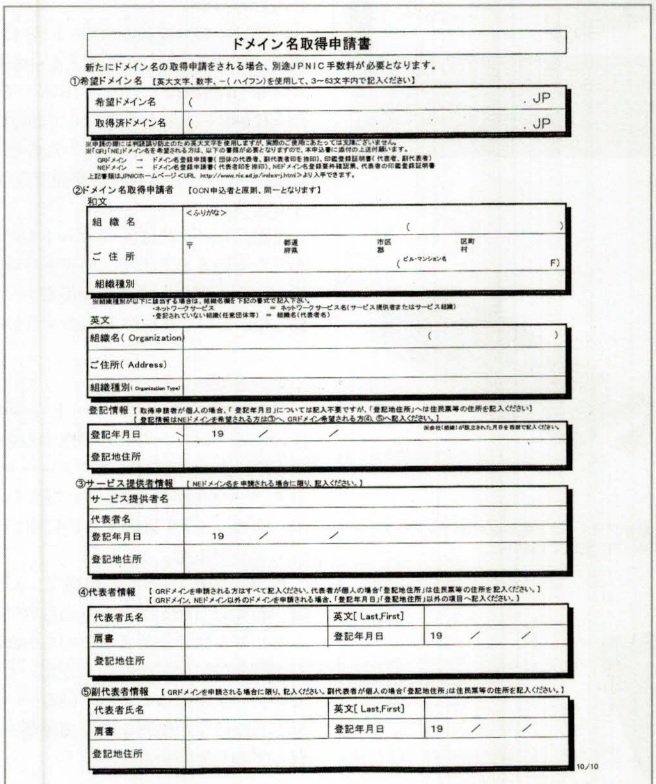
もうひとつはノートパソコンに搭載されているHDDのMTBF(平均故障間隔時間)がデスクトップPC向けHDDよりも短い(つまり故障しやすい)ため，信頼性にやや劣るからだ。デスクトップPCであっても，特殊ハードウェアになりがちな省スペースPCなどは同様な理由で避けたほうがいいかもしれない。

さて，メーカー製パソコンではなく自作パソコンでもいいのだが，その場合，安定性が要求されるので最新パーツとか無名メーカー製の超特価品とかは避けたほうがいい。メジャーなメーカーの「枯れた」スペックのパーツを組み合わせたほうがいい。

たとえば，ソケット7系ならば430TX，PentiumII系ならば440BX，LX，FXといったチップセットを搭載したものがいいだろう。

動かすOSにもよるが，メモリは64MB程度，HDD容量は2G〜4GBでOKだ。ビデオカードは画面が映ればいいので高級な3DアクセラレータやAGP機能はいらない。サウンドカードも不要だ。そして決してオーバークロックなんかしないこと。

メーカー製PC，自作PC，いずれの場合も電源はATXではなく，AT電源のほうがなにかと都合がいい。というのは，AT電源ならば物理的なスイッチングで電源が投入されているため，一時的な停電などのとき，電気の供給が復活した瞬間に再起動がなされるからだ。ATX電源の場合だと，マザーボードへの電源投入は電源スイッチのトリガに反応して行われているため，人間の手でもう一度電源スイッチを押してやらなければならなくなる。もちろん無停電電源装置などでサーバに電気を供給すれば回避できる問題ではあるが……。



ドメイン名取得申請書

新たにドメイン名の取得申請をされる場合、別途JPNIC手数料が必要となります。

①希望ドメイン名 【英大文字、数字、−(ハイフン)を使用して、3〜63文字内で記入ください】

希望ドメイン名	(. JP
取得済ドメイン名	(. JP

※申請の際には判読誤り防止のため英大文字を使用しますが、実際のご使用にあたっては支障ございません。
※「GR」「NE」ドメイン名を希望される方は、以下の書類が必要となりますので、本申込書に添付の上送付願います。
GRドメイン → ドメイン名登録申請書(団体の代表者、副代表者印を捺印)、印鑑登録証明書(代表者、副代表者)
NEドメイン → ドメイン名登録申請書(代表者印を捺印)、NEドメイン名登録要件確認票、代表者の印鑑登録証明書
上記書類はJPNICホームページ<URL http://www.nic.ad.jp/index-j.html>より入手できます。

②ドメイン名取得申請者 【OCN申込者と原則、同一となります】

和文

組織名	<ふりがな> ( )
ご住所	〒 都道府県 市区郡 区町村 (ビル・マンション名 F)
組織種別	

※組織種別が以下に該当する場合は、組織名欄を下記の書式で記入下さい。
・ネットワークサービス ＝ ネットワークサービス名(サービス提供者またはサービス組織)
・登記されていない組織(任意団体等) ＝ 組織名(代表者名)

英文

組織名(Organization)	( )
ご住所(Address)	
組織種別(Organization Type)	

登記情報【取得申請者が個人の場合、「登記年月日」については記入不要ですが、「登記地住所」へは住民票等の住所を記入ください】
【登記情報はNEドメインを希望される方は③へ、GRドメイン希望される方④、⑤へ記入ください。】

登記年月日	19 / / ※会社(組織)が設立された月日を西暦で記入ください。
登記地住所	

③サービス提供者情報 【NEドメイン名を申請される場合に限り、記入ください。】

サービス提供者名	
代表者名	
登記年月日	19 / /
登記地住所	

④代表者情報 【GRドメインを申請される方はすべて記入ください。代表者が個人の場合「登記地住所」は住民票等の住所を記入ください。】
【GRドメイン、NEドメイン以外のドメインを申請される場合、「登記年月日」「登記地住所」以外の項目へ記入ください。】

代表者氏名		英文[Last,First]	
肩書		登記年月日	19 / /
登記地住所			

⑤副代表者情報 【GRドメインを申請される場合に限り、記入ください。副代表者が個人の場合「登記地住所」は住民票等の住所を記入ください。】

代表者氏名		英文[Last,First]	
肩書		登記年月日	19 / /
登記地住所			

10/10

図8

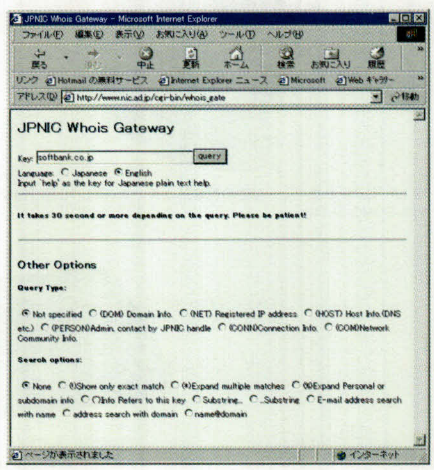


図9

サーバ事故の被害を最小限に抑えるために RAID1システムを導入しよう

Column

今回構築するサイトのように，1台のサーバですべてのサーバ機能をまかなう場合，恐ろしいのはなんといってもサーバクラッシュだ。

そのサーバを復活させないことにはそのサイト全体が死んだままになってしまう。

「こまめにバックアップを取ろう」は「真理のキーワード」なので否定するつもりはまったくないが，サーバの場合はそれも難しい。

そこでおすすめしたいのがRAID1……ディスクミラーリングシステムだ。2台に常に同じ内容を並列に書き込んでいくディスクアレイシステムで，いわばリアルタイムにHDDのバックアップを取りながらサーバを運用していく……というイメージだ。

マシンのクラッシュの代表的なパターンはなんといってもハードディスクの故障によるものだが，2台のHDDが同時にクラッシュすることはまずありえない。万が一，1台のHDDが故障してしまっても，RAID1システムならば，その壊れたHDDを外し，もう1台の正常なHDDから再起動すればクラッシュ寸前の環境が復活できる。

さて，そのRAID1システムだが，これを構築するには2台のHDDとRAIDカードというインタフェイスカードが必要になってくる。

OCNエコノミー程度のサーバならばHDDは4GB以下でも十分だろう。現在IDEの4GB前後のHDDは1台13000円前後。2台買っても3万円以下だ。

問題はRAIDカードだが，PROMISE製の「FastTrak」シリーズという非常に安いRAIDカードが出ているのでこれを利用するといい（写真1）。

「FastTrak」(http://www.promise.com/Products/ideraid/fasttrak.htm)はUltraATA/33対応で実売価格10000円前後。最近ではUltraATA/66に対応した「FastTrak66」(http://www.promise.com/Products/ideraid/ft66page.htm)という製品も発表になり，こちらは2万円前後で販売が開始されている[*1]。

結局，4万円以下の予算でRAID1システムが構築できることになる。

この投資額が安いか高いかの判断は各人に委ねられるが，「クラッシュ時のサーバ復活の手間」を考えるとすれば個人的には安いものだと思うのだが，どうだろうか。

(*1)いずれの製品もDOS/Vパラダイス(http://www.dospara.co.jp/)が正規代理店になっている。



PROMISE FastTrak66。実勢価格は1万6000～1万8000円



単体RAIDドライブTrustGuardの3ドライブ版。SCSI接続だ。IDE版は9万8000円～　㈱エヌエスピー研究所 http://www.nspi.net/



図10　OCNエコノミー対応の代表的なルータNTT-TE製「MN128-SOHO SL11」

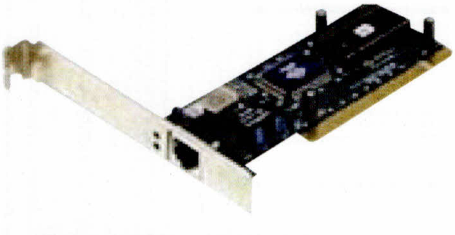


図11　enw8300t.jpg プラネックス製10BASE-Tイーサネットアダプタ「ENW-8300-T」。定価で1980円だ



図12　lghm.jpg 低価格な16ポートハブ。メルコ「LGH-M16」定価7,800円

●用意するもの

・ルータ

ルータはすでに購入してあるだろうか。OCNエコノミー対応ならばなんでもいいのだが，ただ「OCNエコノミー対応」というキーワードだけに注目して買ってくると，OCNエコノミー対応のTA/DSU装置だったりするので注意（笑い話じゃなくてそういう間違いをする人がいる）。必ず「DSU内蔵ルータ」を買うこと（図10）。

・ネットワークカード

いうまでもないことかもしれないが，サーバとして使うパソコン，そしてそのネットワークに参加するすべてのパソコンには，10BASE-T（あるいは100BASE-TX）のネットワークカードを搭載しなければならない。

OCNエコノミー経由でインターネットにアクセスしたいパソコンには，サーバ，クライアントの区別なく，すべてLANカードを搭載する必要があるということだ。

10BASE-TタイプのものならばPCIバス接続タイプでも現在は2000円以下で販売されているので，そうたいした出費にはならないはず（図11）。

・ハブ（イーサネットハブ）

複数のパソコンをOCNエコノミー経由でインターネットに接続させる場合にはイーサネットハブ（通称ハブ）が必要になる。

NTTから自分のところへ引かれたOCNエコノミーの線を自分のところにあるパソコンすべてへ分配するような装置……イメージ的にはそんなところだろうか。

ハブはその分配数（＝ポート数）によって価格が異なってくる。OCNエコノミーの契約でIPアドレスを16個取得した人は16ポート，8個取得した人は8ポートのものを買っておけばいいだろう。

ハブも10BASE-T対応のものならば現在は激安で，16ポートのものが実売で6000円台で買えてしまう。

なお，ルータには3，4ポート分のハブ機能が備わったものもあるので，ネットワークに参加させるパソコンが少なくて，今後もパソコンの数が増加しないという確証があればハブは不要だ（図12）。

・ネットワークケーブル

10BASE-Tタイプのネットワークケーブルを，接続するパソコンの数分用意する必要がある。ネットワークケーブルにはストレートケーブルとクロスケーブルがあるが，ハブとパソコン（ネットワークカード）を接続するのに用いるのはストレートケーブルだ。

なお，10BASE-Tの規格上，ハブとパソコン間の距離は100m以内に決められている。

ケーブルはさまざまな色のものが販売されているが機能的には差がない。ただ，どのパソコンがどのケーブルにつながっている……というのは把握できたほうが便利なので何種類かの色のものを買っておくといい。

配線してみよう

OCNエコノミーは専用線といっても海底ケーブルのようなぶっとい線が部屋の中に持ち込まれるようなことはなく，6芯メタル線の，外見はただの電話線だ。実際ISDN回線と同じケーブルだったりする。

屋外からきているこのケーブルをルータに接続する。

ルータには数ポートの10BASE-Tのハブ機能が備わっていると思うので，ここにネットワークケーブルを接続し，ケーブルの反対側を別途購入したハブと接続する。

ルータとハブを接続する際に確認しておきたいのはハブ側に装備されている端子のタイプだ。

ハブ側の端子の中に「UPLINK端子」，もしくは「カスケード接続端子」というのがあるならば，ルータとここをストレートケーブルで接続する。もし，買ってきたハブにそうした端子がないならば，ルータとハブをクロスケーブルで接続する必要がある（有名メーカーのハブならば，たいていはそうした端子がついているとは思うが）。

今後ハブを増設する場合も同様で，ハブAの普通の端子と，ハブBのUPLINK端子をストレートケーブルで接続すればいい。

配線の概念図を図13に示しておこう。

サーバとそれ以外の普通のPC（便宜上，クライアントと呼ぶ）とは配線上は同列だ。サーバだからといって特別製のケーブルで接続したりする必要はない。

サイト構築編～ソフトウェア編

サーバOSはなににするか

配線が終わったら，いよいよソフトウェアの設定に移ろう。

まずは，自分のネットワークの要ともいうべきサーバを動かさなければなにも始まらない。まずはサーバとなるパソコンで動かすOSを決定しよう。

最近では導入コストの安くてサーバとして実績のある，フリーUnix系が人気なのだが，ここではWindowsベースのサイトを構築する例を示してみることにする。

結局，最近のPCユーザーの場合，Windowsの操作に慣れてしまっているし，アップデート情報やセキュリティ情報など，なにかと新しい情報が「日本語で入ってくる」メリットは見逃せない。ただし，その分コストはちょっと高くなる。

「Windows2000 Server」は実売価格で180,000円前後もする。学生(小中高生も可能)や学校関係者ならばまったく同じ製品で価格が77,000円と半額以下のアカデミーパックが購入できる。また，以下の競合製品から乗り換える場合には88,300円の「乗り換えアップグレードパッケージ」を利用することも可能だ。

なお，機能的にさほど差がない「Windows 2000 Professional」ならもっと安くできるのだが，ライセンス的に多少きな臭いところもある。一応，Windows NTではNT Workstation（Windows 2000 Professionalに相当）では明確にサーバ製品の使用は禁じられていたのだが，Windows 2000ではインターネット用のサーバなら構築しても問題がない（ただし，接続人数は最大で10人に制限する必要がある）という見解となっているようだ。

普通の人が購入するのはProfessionalのほうだろうが，サーバ機能の充実度とサポートの問題から，以下では2000Serverを主として説明を行う。だいたいの手順はProfessionalでも同じだが，DNSなど一部ソフトウェアは付属していないので，フリーウェアなどで代替する必要がある（使い勝手はあまりよくない）。が，その気になれば必要なツールはすべてフリーウェアで揃っているので，Windows2000 Professionalをベースに安く上げることもできなくはない。ただし，IISを使わないと，10人という同時接続制限をかけることが難しかったり，それならいっそPC Unixベースでやったほうが楽かもしれないという気もしないではないのだが。ちなみに，Windows98をサーバにする場合には接続制限はまったくない。安定性やセキュリティなどで問題があるが，Windows98ベースというのも不可能というわけではない（Oh!XのネットワークゲームWebサーバはDirectXの関係でWindows98ベースで動作していたが，数カ月の連続運転でも特に問題は発生していなかった）。

・乗り換え アップグレード対象製品

Novell : NetWare，IntranetWare，IntranetWare for Small Business
Banyan Vines
IBM : LAN Server，OS/2
Microsoft LAN Manager
DEC PATHWORKS
Artisoft LANtastic
SCO : Xenix，UNIX，OpenServer，UnixWare
Sun : Solaris，Solaris X86，SunOS
Hewlett-Packard : HP-UX
IBM : AIX
Digital : Ultrix，OSF/1，UNIX
SGI : Irix

Windows2000 Serverをインストールしよう

Windows2000 Serverのインストール方法は2とおり。セットアップFDから起動してインストールする方法と，CD-ROMから直接起動してインストールする方法だ。CD-ROMからインストールする場合はPCのBIOSのBootドライブの設定で「CD-ROM」が選択できないとダメ。最近のPCではほとんどこの設定が可能だが，古いPCだとこれができないものが多い。どちらの方法でもインストールされるWindows2000 Serverに変わりはないが，CD-ROM起動からのほうが速いしFDの交換がいらないので楽だ。

インストールは基本的にメッセージに従って行っていけばいい。

インストールの初期段階で注意したいのはファイルシステムの選択だ。FATとNTFSのどちらのファイルシステムにするか選択を迫られる。NTFSのほうがセキュリティ機能が豊富で，またフォルダ単位の圧縮機能なども備わっており，なにかと便利なので，特別な事情がない限りはNTFSを選択しよう。

Windows2000 Serverをインストールしよう～セットアップウィザード

インストールの最終段階ではセットアップウィザードが起動し，対話的にインストールが進められる。

今回の設定例では，このサーバをDNSサーバとして機能させるものとして話を進めていく。

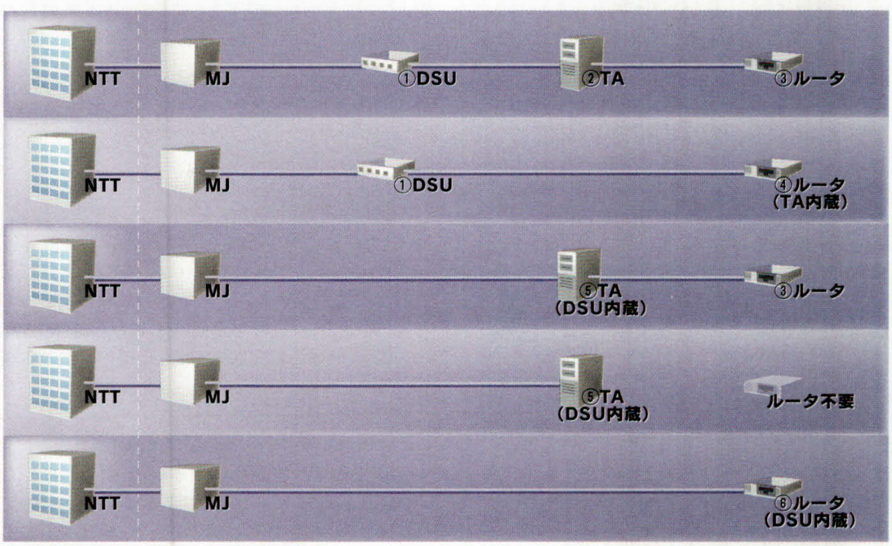


図13　ネットワークの構成

●地域

表示形式やキーボードレイアウトの設定を行う項目だが，日本国内で使用する一般ユーザーの場合，なんの設定変更もせず飛ばしてしまってかまわない。[次へ]ボタンを速やかに押そう。

●ソフトウェアの個人用設定

次に「名前」と「組織名」を聞いてくるのでこれを入力する。「名前」には自分の名前，あるいはネットワーク管理者の名前を記入し，「組織名」はグループ名や企業名，部署名などを入力する。個人でOCNエコノミーを契約しているのならば取得したドメイン名でもいいだろう。

●ライセンスモード

ライセンスモードの選択を行う。「同時使用ユーザー数」と「接続クライアント数」のどちらかを選ぶことになる。

規模が大きいネットワークで複数のサーバが存在し，購入したWindows2000 Serverのクライアントライセンス数が多い場合は「接続クライアント数」モードでいいが，普通は「同時使用ユーザー数」を選択したほうがいい。

「同時使用ユーザー数」を選択した場合には，併せて「同時接続」の数値も入力する必要がある。ここは購入したWindows2000 Serverに付属してきたクライアントアクセスライセンス数を入力する。パッケージにライセンス数が書かれているので確認してみよう。標準的なパッケージである「Windows2000 Server」のパッケージは「5クライアントアクセスライセンス」になっているはずだ。

●コンピュータ名とAdministratorのパスワード

続いてコンピュータ名を入力する。なんでもいいが，いかにもサーバらしい名前をつけたほうがあとあとネットワークに参加するPCが増加したときにわかりやすい。

たとえばメールサーバならば「MAIL」，WWWサーバならば「WWW」，DNSサーバならば「DNS」といった具合だ。今回のような1台のPCですべてのサーバ機能をまかなう方針であれば無難に「SERVER」でいいだろう。

ここでは同時にサーバ管理者がログインする際のパスワードの設定を求められる。サーバ管理者というのは今回のようなケースでは自分に他ならないので自分で適当に決定する。ただし忘れた場合や流出した場合には最悪全システムの設定のやり直しになるのでその管理には気をつけること。

●Windows2000 コンポーネント

インストール時に組み込むコンポーネントの設定を行う。

今回の例ではDNSはWindows2000 Server付属のものを使用するので，「ネットワークサービス」の中の「ドメインネームサービス(DNS)」をチェックオンにしておくこと。その他の項目については基本的にはデフォルトのままでいい。

●日付と時刻の設定

特に悩むところもないので適当に設定して次へ進む。

●ネットワークの設定

「標準設定」か「カスタム設定」かの選択を要求される。特に変わったプロトコルを導入する予定がなければ「標準設定」を選択してかまわない。

なお，「ネットワークプロトコル」にしろ「ネットワークサービス」にしろ，Windows2000 Serverインストール後はいつでも追加削除できるようになっている。

●ワークグループまたはドメイン名

コンピュータをドメインに参加させるかどうかを決定する。といっても現在セットアップしている環境にネットワークはまだ存在していないので「このコンピュータはネットワーク上にないか，ドメインのないネットワークに接続している」を選び，ワークグループ名は適当に決めるか，デフォルトの「WORKGROUP」のままにしておく。

この設定もWindows2000 Serverインストール完了後，あとで設定変更が可能なので安心してほしい。

●ローカルエリア接続のプロパティ

このあとネットワークの設定をインストーラが自動的に行ってくれる。

その際，ルータ側のDHCP機能(動的にIPアドレスをクライアントに割り当てる仕組み)がONになっていると「DNSの安定した操作を行うには静的なIPアドレスだけを使用してください」という注意を促すメッセージを出し，「ローカルエリア接続のプロパティ」が開く場合がある。この時点で後述するサーバマシンのIPアドレスなどを設定してもいいが，そうした設定は後回しでもかまわないので[OK]ボタンを押してインストールをそのまま先に進めてしまってもかまわない。

なお，ここまでの設定を完了し，しばらくするとリセットがかかりマシンは再起動する。

●サーバの構成

サーバの種類を「このサーバはネットワーク内にある唯一のサーバです」「ネットワーク内にすでに1つ以上のサーバがあります」「このサーバをあとで構成します」の3項目から選択するように迫られる。

今回の例ではサーバは1台にするので最初の選択肢を選び[次へ]ボタンを押す。

このあと「ActiveDirectory」「DHCP」「DNS」といったネットワーク用語の解説が見られる画面に進むが，特に興味がなければここも[次へ]ボタンを押して先へ進んでしまっていい。

●ドメイン名をどのような名前にしますか?

続いて出て来るのがドメイン名関係の入力画面だ。

ややこしいのだが，ここでいっているドメイン名はJPNICより取得したドメイン名のことではなく「自分の持っているパソコンで形成するネットワークの名前」のことをいっている。好きな言葉など，適当に決めてもいいのだが，取得したドメイン名の主要名を入れておくのが一般的だろう。たとえば今回「SAMPLE.CO.JP」というドメイン名を取得したのであれば「SAMPLE」といった名前がいい。ここでは取得したドメインとこのドメインを区別して話をわかりやすくするために「DOMAIN」を与えておくことにする。

●インターネットに登録されているドメイン名

これはJPNICより取得したドメイン名をそのまま入力すればいい。「SAMPLE.CO.JP」であればこれをそのまま入力する(図14)。

ネットワーク情報の設定

インストールが完了したらIPアドレスの入力を行う。

これはNTT側から送られてきた書類に記載されている情報を参考に，正しい値を入力しなければならない。

●IPアドレスの入力

工事のあと，NTTから「申し込み手続き完了」の封書が届くが，この中に含まれていた「お客様ドメイン名」などが記載されている書類を見てほしい。ここに「IPアドレス」という項目があるはずだ。ここに4つの数字がピリオド(.)で区切られて記載されていると思う。また最後の数字の右側には「/28」のような記載があると思う。

本稿では話をわかりやすくするために

お客さまドメイン名 sample.co.jp

IPアドレス 200.140.160.176/28

と記載されていたものとして話を進める。この値は，いわば電話でいえば電話番号に相当するもので，契約者ごとに違うのでそのあたりはご了承い

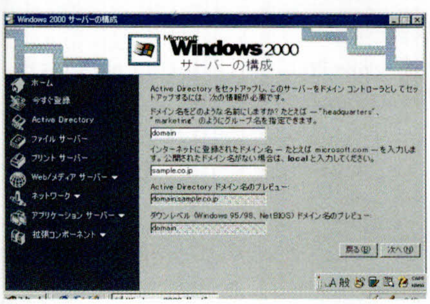
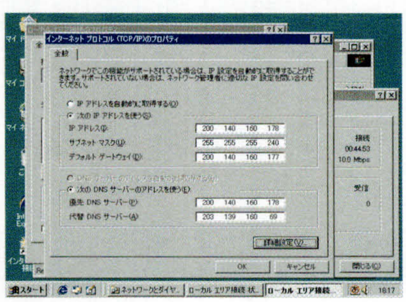


図14


図15

ただきたい。

まず，コントロールパネルの「ネットワークとダイアルアップ接続」を開き，「ローカルエリア接続」のアイコンをクリックする。この操作で開かれるウィンドウ内にある「プロパティ」ボタンを押すと，現在組み込まれているネットワークコンポーネントが表示されるはずだ。

表示されたコンポーネントのうち「インターネットプロトコル（TCP/IP）」をダブルクリックしてプロパティを開くと図15のようなウィンドウが現れる。

サーバはIPアドレスを静的に与えるのが一般的なので「次のIPアドレスを使う」の項目を選択し，数値記入欄に正しい値を入力していくことになる。

今回の例では16個取得しているので，

200.140.160.176～200.140.160.191

が使えるわけだが，

200.140.160.176はネットワークアドレス

200.140.160.191はブロードキャストアドレス

としてリザーブされ，さらに，

200.140.160.177はルータのIPアドレス

として使用することになるため（詳しくは後述），結局ここでは，使えるIPアドレスのもっとも若い値である，

200.140.160.178

を入力する。

この設定により，このパソコンのIPアドレスは固定されることとなり，このアドレス値でこのパソコンがインターネット上から参照されることになる。

次の「サブネットマスク」はIPアドレスを16個取得している場合は，

255.255.255.240

とする。取得したIPアドレスが8個の場合は最後の数値を「248」とする（コラム「サブネットマスクとは」参照）。

最下段の「デフォルトゲートウェイ」にはルータに設定すべきIPアドレスを入力する（後述）。具体的には取得したIPアドレスのうちもっとも若い数値に+1すればいいので，今回の例では，

200.140.160.177

となる。

ちなみに，「サブネットマスク」「デフォルトゲートウェイ」の2つは，今後，自分のネットワークに参加するすべてのパソコン（クライアント）で共通の設定値となるので覚えておこう。

●DNSの設定

項目「次のDNSサーバのアドレスを使う」を選択し，今度はここへDNSサーバのIPアドレスも入力する。

今回の例ではいまセットアップしているこのサーバをDNSサーバとする予定なので，このパソコンに設定したIPアドレス，

200.140.160.178

を「優先DNSサーバ」（いわゆるプライマリDNSサーバ）の欄へ入力すればいい。

「代替DNSサーバ」（いわゆるセカンダリDNSサーバ）への欄にはもう1個のDNSサーバのIPアドレスを入力する。セカンダリDNSサーバはOCN（NTT）側にあり，すでに決まっているのでこれを入力する。NTTから送られてきた書類に，「セカンダリDNS IPアドレス」という項目があると思うのでここの値を入力すればいい。

なお，東日本の契約者は，

203.139.160.69

西日本の契約者は，

203.139.161.37

になっていることが多いと思うが，違う場合もあるので各自確認してほしい。

ルータの設定

さて，いまの状態ではただ，サーバOSをインストールしただけで，ネットワークのサーバとして機能していない。今度はルータに対して正しい設定を行う必要があるのだ。

ルータの設定方法はルータごとに異なるのでマニュアルを参照してもらうしかない。しかし，それではあまりにもつれないので，本稿では冒頭で紹介したNTT-TE販売のMN128-SOHOシリーズ（以下MN128と略記する）の設定方法を紹介する。機種が違っても少しは参考になると思う。

●Webブラウザを起動する

MN128の内部パラメータはネットワークを介して行える。サーバマシンとMN128はちゃんとネットワークケーブルで接続されているだろうか。

接続されているならば，InternetExplorerなどのWebブラウザを起動し，

http://setup.mn128-soho

あるいは工場出荷状態のMN128のIPアドレスである，

http://192.168.0.1

を入力しよう。図16のような設定メニュー画面が現れるはずだ（MN128-SOHOシリーズは何種類かあるので機種によっては画面が違う）。

あまりないとは思うが，中古のMN128を買ってきた場合はそうしたIPアドレスや，装置名が変更されている可能性がある。こういうときは慌てず騒がず，工場出荷状態に戻せばいい。やり方はMN128のオンラインマニュアル（CD-ROM版）の「保守」の項を参照してほしい。なお，初期化はパソコンがなくてもアナログ電話機を使っても行える。

以降，MN128の設定パラメータの変更はこのWebブラウザを用いる方法で行うことになる。

●本体設定

まずは「本体設定」（図17）から行ってみよう。

まず「ルータ名称」だが，特に変更する必要はないのでそのままでいい。

「設定する時刻と日付」については現在の正しい時刻と日付を入力する。

「ログインパスワード」は，このMN128の設定を他人が変えられないようにするためのセキュリティ機構だ。これは絶対に設定しておくこと。

ルータはインターネットから見えるので，IPアドレスさえわかってしまえば他人からなんの苦労もなくアクセスされてしまう。悪意のあるユーザーにアクセスされた場合，ルータの内部パラメータを不当に書き換えられる可能性もあり，そのときは自分のサイトがダウンさせられてしまうこともある。

なお，設定したパスワードは忘れないように絶対どこかにメモを取っておくこと。忘れてしまった場合は，ルータを一度工場出荷状態に戻し，全部の設定を最初から行わなくてはならなくなる。

●IPアドレスの設定

次にルータの設定でもっとも重要な設定を行うことにしよう。それがIPアドレスの設定だ。

MN128シリーズでは「ルータ設定」の項目の「IP設定」というメニューを選択する。

すると図18のような画面が現れるはずだ。

まず入力すべきなのは本体の「IPアドレス/サブネットマスク長」。

ここでルータのIPアドレスを設定することになる。通常は取得した「IPアドレス」の4番目の値に1を加えた値を書いておけばいい。

今回の例では，

200.140.160.177

ということになる。気づいた人もいると思うが，これはサーバマシンにWindows2000をインストールしたときに「デフォルトゲートウェイ」として入力したアドレス値と一緒だ。そう，自分のサイトにおいて，ゲートウェイを務めるのはこのルータなのだ。

「サブネットマスク長」は取得したIPアドレスが16個の場合は，

/28

を最後尾に記載する。8個の場合は「/29」になる。

このほかの項目は，とりあえずデフォルトのままでいい。

入力が完了したら［設定］ボタンを押そう。

なお，今後，このルータの設定画面を呼び出す場合には，Webブラウザでここで設定したIPアドレスを入力することになる。今回の例ならば

http://200.140.160.177

を入力すればいい。

「本体設定」にてパスワードを施しているので，直接設定画面は開かれず，最初は認証画面が出

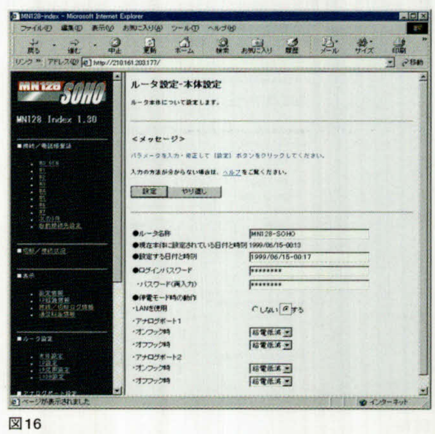


図16

てくるはずだ(図19)。ここで「ユーザー名」は「Admin」，「パスワード」には自分で設定したパスワードを入力すれば，設定画面が開かれる。

セキュリティの確保

●セキュリティはどうする?

OCNエコノミーは低価格専用線といっても，実際専用線なので常時インターネットに接続されることになる。これはすなわち自分のサイトがインターネットに365日24時間さらされることになるわけで，悪意あるユーザーからのサイト・アタックの可能性も否定できない。

本来，高度なセキュリティを確保するのであればプロキシ(代理)サーバなどを立て，自分のLANとインターネットを分断してしまう方法がおすすめなのだが，これ用にもう1台PCを用意することになるのでコストが余計にかかる。

また，プロキシサーバは導入した場合，ネットワーク対戦ゲームをはじめとした一部のインターネットコンテンツを利用できない場合がある。インターネットの使用形態や求めるセキュリティのレベルなどをよく吟味して，このあたりは決定すべきだ。まぁ，私個人としては，プロキシサーバはおいおい金銭的な余裕ができてきたら用意するとして，最初はルータ側のフィルタ設定でセキュリティを確保すればいいと思う。個人ベースやSOHOクラスの企業であればプロキシサーバがなくても不便を感じることはまずない。

どうしてもプロキシサーバを導入したいというのであれば，ネットワークカードを2枚挿したPCを1台用意しなければならない。そして，これにWindows2000 Serverを最小限のネットワークコンポーネントでインストールし，さらにここにプロキシサーバソフトをインストールする(図20)。

WindowsNT/2000向けのプロキシサーバソフトとしてはマイクロソフトが出している「Microsoft Proxy Server 2.0」(http://www.microsoft.com/japan/BackOffice/Proxy/default.htm)があるが実売価格が10万円前後と結構高い(アカデミパックはその半額くらい?)。このほか，サードパーティの製品としてはブレーンの「PROXY 2000」(http://www.brain-tokyo.com/proxy2000/)という製品がある。こちらは5ユーザー向けが実売価格で1万円前後とかなりリーズナブルだ。

●IP応用設定～フィルタの設定

パケットフィルタリングとは，インターネット側からやってくる通信パケットのうち「どういったものを通してどういったものをブロックするか」を判断する機構のことだ。MN128をはじめとしたルータにはこうした機構が標準で搭載されている。今回の例では，セキュリティ確保は，このルータ側のパケットフィルタリングのみで実現することにする。

インターネットは知ってのとおりTCP/IPベースのネットワークだ。さまざまなサービスが，それぞれ異なるTCP(Transmission Control Protocol)，UDP(User Datagram Protocol)ポート番号を使っているのだが，パケットフィルタリングとは具体的には「どのポート番号が使われているパケットを通して，どれをブロックするか」という設定を行うことをいう。

たとえばメールサービスのSMTPはポート番号25だし，DNSは53，WWW(HTTP)は80，FTPは20，21，POP3は110，113といった具合になっており，たとえば，こうした主要サービスのみを使う……という場合，上記以外のポート番号を全部ブロックしてしまうような設定にすればいい。ただし，その場合，インターネットを利用した多くのアプリケーション(音楽配信や対戦ゲームやその他いろいろ)が使えなくなってしまうためあまり一般的ではない。

「じゃあなにを通してなにをブロックすればいいのよ」ということになるわけだが，これは各ユーザーのOCNエコノミーの利用形態に左右されるのでなかなか「こうすればいい」というアドバイスがしづらい。そこでここでは，MN128のオンラインマニュアルで奨励している設定と，世間一般で慣例的に用いられている設定の両方を考慮した例を示すことにする。

まず，MN128の設定メニューで「IP応用設定」を選び，図21を出そう。フィルタの設定はこの画面の中の「●オプション」というテキストボックスの中に行うことになる。

この記述言語の文法はMN128の場合は，リスト2(ipfiltering.txt)のようになっている(一般的にはどのルータでもだいたい同じ)。

さて，今回設定例として示すのはリスト3(ipf_sample.txt)だ。なにをしているかざっと説明しておこう。

TCP，UDPのポート番号135,137-139はMicrosoft Windows Networkで用いられるプロトコルだが，これを全面的にブロックしている。これはネットワーク内の共有資源がインターネット側から不正に使用されるのを防止する役目を果たしている。

そしてfilter番号10，11ではtelnetによるルータのアクセスを禁止している。ここに記載されているIPアドレスは「IPアドレスの設定」で設定したルータのIPアドレスだ。telnetとはサーバを遠隔操作するソフトウェアだが，この2行でこれによるルータの操作を禁止していることになる。

これを正しく入力したら，画面上の[設定]ボタンを押す。

MN128の場合，フィルタ定義文は最大で32個まで設定できるので，ここで示したもの以外にもあと10数行記述できることになる。ここで示したようなポート番号のブロック以外に，LAN内の特定ユーザーのみをインターネットにアクセス

サブネットマスクとは　Column

サブネットマスクとは，ネットワーク内のIPアドレスとANDを取るとそのサブネットのIPアドレスが得られるようなマスクビット列のことである。

「/28」と表記した場合，「マスクビット列を最上位ビットから1を28個分で表現する」…という意味になる。最上位ビットから1が28個続くということは，2進表現で，

11111111_11111111_11111111_11110000

ということになるわけだが，これを8ビット単位で10進表記すると

255 255 255 240

ということになる。サブネットマスクをこちらの方式で表現することも多い。

今回の例で，サーバマシンのIPアドレス設定時にこの4つの数値を「サブネットマスク」として入力したことがあったと思うが，あのときはこちらの8ビット区切りの10進表記で値を入力したわけだ。

今回の例ではIPアドレスを16個取得しているので，マスク除去されるべきは最下位4ビットだ。だから，32－4＝28，サブネットマスク長が「28」なのだ。

取得したIPアドレスが8個の場合は，マスク除去されるべきビットは最下位3ビットとなる。だから32－3＝29，サブネットマスク長は「29」となる。

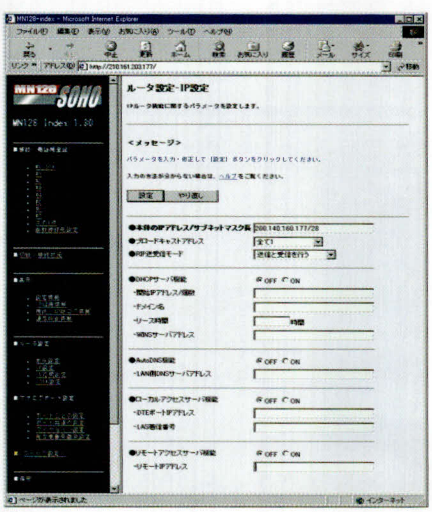


図18



図19

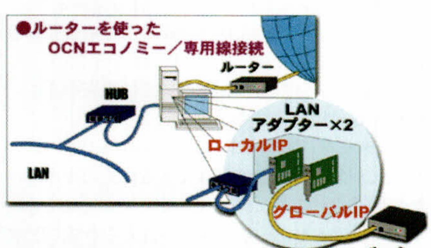


図20　PROXY2000のカタログからの抜粋

可能にしたり，特定のIPアドレスのユーザーを自分たちのサイトから排除したりするような「アクセス制限設定」のようなこともできる。いろいろと研究してみよう。

以上でルータの設定の基本的な部分は終わりだ。

DNSサーバソフトのインストール

さて，いま一度，サーバのほうに話を戻そう。

サーバマシンにはWindows2000 Serverをインストールしてあり，さらに基本的なネットワークの設定も完了しているのだが，このままではまだ不十分だ。続いて自分のネットワーク内にDNSサーバを持つ必要があるのだ。

DNSとは簡単にいうとインターネット上に散らばるさまざまなサーバの名前とIPアドレスを対応づけてくれる機構のことで，この機能を果たすサーバがDNSサーバということになる。これはネットワーク内に絶対必要となるので，必ず準備しなくてはならない。

今回の例ではDNSサーバとしての機能を最初にセットアップしたサーバマシンに持たせることにする。

●DNSサーバソフトウェアはどれにする?

Windows2000 Serverには標準で「ドメインネームシステム(DNS)」が付属してくる(無料)。このほか，無料のDNSサーバソフトとしてはSoftware.comの「BIND for NT」があり，こちらは同社のWebサイト(http://www.software.com/)にて最新版がダウンロード可能だ。

どちらでもいいのだが，今回はWindows2000 Serverに標準付属している「ドメインネームシステム」を使うことにする。

●「ドメインネームシステム(DNS)」をインストールしよう

Windows2000 Serverインストール時に同時にインストールした場合は不要だ。忘れてしまった場合はコントロールパネルの「アプリケーションの追加と削除」を開き，「Windowsコンポーネントの追加と削除」を選択して「Windowsコンポーネントウィザード」からインストールを行う。この中のコンポーネントリストから「ネットワークサービス」を探し，「詳細」ボタンを押して「ドメインネームシステム(DNS)」をチェックオン，[OK]ボタンを押せばあとは自動的にインストールが行われる。

DNSの設定

いま一度，確認しておくが，現在例として設定している，サイト，サーバの情報は下記のようになっている。

取得したIPアドレス(の先頭)：200.140.160.176
取得したIPアドレスの個数：16個
取得したドメイン名：sample.co.jp
サブネットマスク：255.255.255.240
デフォルトゲートウェイ
(ルータのIPアドレス)：200.140.160.177
サーバホスト名：SERVER
プライマリDNSサーバIPアドレス：200.140.160.178

これを前提として話を進めていくことにする。

●「新しいゾーン」ウィザード

システムが再起動したら，再びAdministratorでログインし，今度は「DNSマネージャ」を起動しよう(プログラム名は「DNS」となっている)。

[スタート]メニューから[プログラム]-[管理ツール(共通)]-[DNS]とたどっていけば起動できる。

起動したらウィンドウ左側の「DNS」のアイコンを選択し，現在セットアップしているサーバの名前を出す(今回の例では「SERVER」が出てくる)(図22)。

次にウィンドウ右側に出ているSERVERを右クリックしてメニューを出し「新しいゾーン」を選択する。すると「新しいゾーン」ウィザードが起動するので順次必要な情報を入力していく。

●ゾーンの種類

ゾーンの種類を「ActiveDirectory」「標準プライマリ」「標準セカンダリ」の3つのうちから選択する。ここではデフォルトでもある「標準プライマリ」を選択する(図23)。

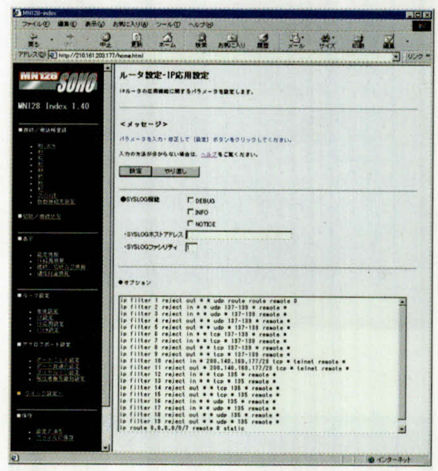


図21

リスト2　ipfiltering.txt

```
ip filter ... IPフィルタの登録

書式：ip filter {fnumber type dir srcaddr dstaddr
 protocol srcport dstport interface [rnumber]}
    fnumber     ... フィルタ番号 [1..32]
    type        ... フィルタタイプ
        pass        ... 一致すれば通す
        reject      ... 一致すれば破棄する
        restrict    ... 回線が接続されている場合だけ通す
    dir         ... 方向
        in          ... 受信時にフィルタリングする
        out         ... 送信時にフィルタリングする
    srcaddr     ... 送信元アドレス[/ネットマスク|-範囲指定] ('*'はすべて)
    dstaddr     ... 送信先アドレス[/ネットマスク|-範囲指定] ('*'はすべて)
    protocol    ... プロトコル番号，またはニーモニック
        ニーモニック... 'udp', 'tcp', 'tcpest', 'tcpfin', 'icmp'
        ('*'はすべて，'tcpest'はSYN, 'tcpfin'はFIN/RSTパケットを対象)
    srcport     ... 送信元ポート番号，またはニーモニック
                    ('*'はすべて，範囲指定は'-'で区切って入力)
        ニーモニック... 'ftp', 'ftpdata', 'telnet', 'smtp', 'www',
                    'pop3', 'sunrpc', 'nntp', 'ntp', 'login',
                    'pptp', 'domain', 'route'
    dstport     ... 送信先ポート番号，またはニーモニック
                    ('*'はすべて，範囲指定は'-'で区切って入力)
    interface   ... 'local'または'remote'
    rnumber     ... 相手先番号 [0..15] ('remote'の場合，'*'はすべての相手先)
```

リスト3　ipf_sample.txt

```
ip filter 1 reject out * * udp route route remote 0
ip filter 2 reject in * * udp 137-139 * remote *
ip filter 3 reject in * * udp * 137-139 remote *
ip filter 4 reject out * * udp 137-139 * remote *
ip filter 5 reject out * * udp * 137-139 remote *
ip filter 6 reject in * * tcp 137-139 * remote *
ip filter 7 reject in * * tcp * 137-139 remote *
ip filter 8 reject out * * tcp 137-139 * remote *
ip filter 9 reject out * * tcp * 137-139 remote *
ip filter 10 reject in * 200.140.160.177/28 tcp * telnet remote *
ip filter 11 reject out * 200.140.160.177/28 tcp * telnet remote *
ip filter 12 reject in * * tcp 135 * remote *
ip filter 13 reject in * * tcp * 135 remote *
ip filter 14 reject out * * tcp 135 * remote *
ip filter 15 reject out * * tcp * 135 remote *
ip filter 16 reject in * * udp 135 * remote *
ip filter 17 reject in * * udp * 135 remote *
ip filter 18 reject out * * udp 135 * remote *
ip filter 19 reject out * * udp * 135 remote *
ip route 0.0.0.0/0/7 remote 0 static
```

●前方または逆引き参照ゾーン
作成する参照ゾーンの種類を選択する（図24）。まずは「前方参照ゾーン」を選択する。

●ゾーン名
ゾーン名を入力する。まず，取得したドメイン名を入力する。ここでは「sample.co.jp」と入力する（図25）。

●ゾーンファイル
先ほど入力したドメイン名に".dns"が追加された文字列が入力欄に表示されていると思うので，このままにして［次へ］ボタンを押して先へ進む（図26）。

Administratorとは
Column

ネットワークコンポーネントの再構成のようなシステムに多大な影響を及ぼすような操作を行うときには必ず「Administrator」アカウントでログインすること。
ここで「Administratorアカウントとはなにか」について，Windows2000/NTをよく知らないユーザーに簡単に説明しておくとしよう。
Windows2000をはじめとするNT系Windowsではそのマシンを使う際に，まずユーザー認証を行わなければならないのである。そのマシン，あるいはそのネットワークに対してもっともハイレベルな設定変更権を持っているのがAdministratorというユーザーなのだ。
なんのことはない，この場合，自分がそうなのだ。
ちなみにWindowsNT系環境で，新しいハードウェアをインストールしたり，ネットワークの設定を変更したり，はたまたシステム情報の更新を必要とするようなソフトウェアをインストールしたりするときもAdministratorアカウントでログインする必要がある。

●新しいゾーンウィザードの完了
ここで［完了］ボタンを押せばゾーンファイルが作成される（図27）。

＊　＊　＊

同様の手順で別のゾーンについても設定を行う。

●ゾーンの種類
「標準プライマリ」を選択。

●前方または逆引き参照ゾーン
今度は「逆引き参照ゾーン」を選択する（図28）。

●逆引き参照ゾーン
ネットワークIDのところに「127.0.0」を入力して次へ進む（図29）。

●ゾーンファイル
入力したIDが逆転され「in-addr.arpa.dns」の文字列が付加された「0.0.127.in-addr.arpa.dns」という名前でファイルを作成する旨が表示されるのでこれを承諾する（図30）。
このあと［完了］ボタンを押せばこの名前でゾーンファイルが作成される。

＊　＊　＊

ウィザードを使っての設定はこれで最後だ。

●ゾーンの種類
「標準プライマリ」を選択。

●前方または逆引き参照ゾーン
「逆引き参照ゾーン」を選択する。

●逆引き参照ゾーン
今度は「逆引き参照ゾーンの名前」の欄に所得したIPアドレスのもっとも若いものを逆順に入力し，末尾に「.in-addr.arpa」を付加した文字列を入力する。今回の例ではもっとも若いIPアドレスは「200.140.160.176」なので「176.160.140.200.in-addr.arpa」を入力することになる（図31）。

●ゾーンファイル
入力した文字列に「.dns」が付加された「176.160.140.200.in-addr.arpa.dns」という名前でファイルを作成する旨が表示されるのでこれを承諾する（図32）。
このあと［完了］ボタンを押せばこの名前でゾーンファイルが作成される。

＊　＊　＊

ここまでの設定を完了するとDNSマネージャは図33のようになっているはずだ。

●エイリアスの設定を行う
今回の例ではひとつのサーバでDNS，WWW，メールの3つのサーバ機能を担当させる方針で作業を進めている。
いまいろいろと設定を施しているサーバのホスト名は「SERVER」になっているので，たとえばインターネットからこのサーバを参照する場合には，

server.domain.sample.co.jp

と名前を指定することになる[*1]。
ところが，WWWサーバは，

www.sample.co.jp

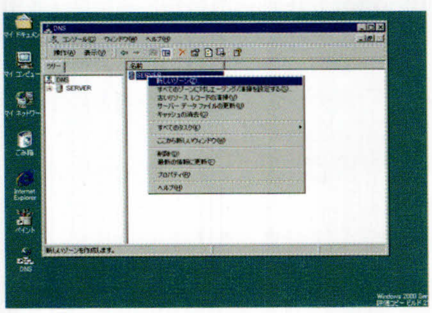
として指定するのが普通だし，メールサーバは，

mail.sample.co.jp

として指定できたほうがわかりやすい。
そこで，こうした名前が用いられたときにも正しく，

server.domain.sample.co.jp

が参照されるような仕組みを作ってやる必要がある。
それではその設定をすることにしよう。


図22

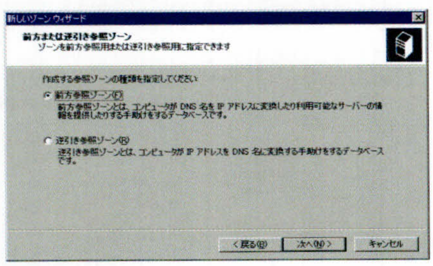


図24

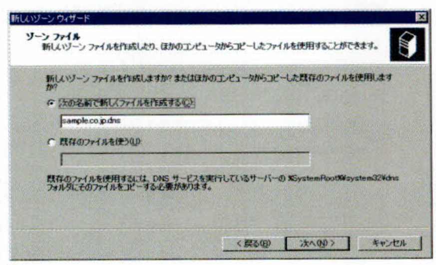


図26

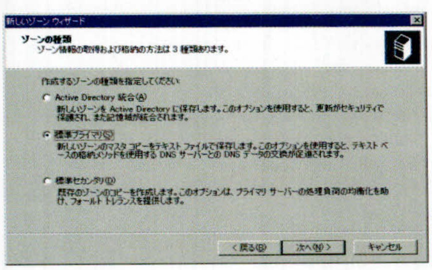


図23

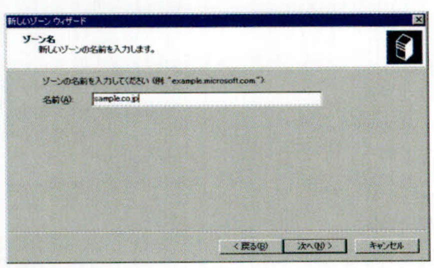


図25



図27

まずは「WWW」という名前を「server.sample.co.jp」に割り当てる手続きをやってみよう。

まず，先ほどの図33でウィンドウ左側の「sample.co.jp」を選択した状態で右クリックしてメニューを出し「新しいエイリアス」を選択する。すると図34のようなウィンドウが表示されるので，ここに必要事項を記入していく。

まず，当たり前だが「エイリアス名」は「WWW」となる。

「ターゲットホスト用の完全修飾名」は関連づけるホスト名，ここでは「server.domain.sample.co.jp」ということになる。

まったく同様の手順で「MAIL」の別名定義も行おう。

「エイリアス名」は「MAIL」，「ターゲットホスト用の完全修飾名」はここでも「server.domain.sample.co.jp」だ。

いうまでもないことかもしれないが，WWWサーバやメールサーバを別のパソコンに担当させる場合はこの手順は不要だ。

(※1) Windows2000serverではホスト名(コンピュータ名)がインストール時にしていた，「コンピュータ名.ドメイン名」で表されるようになっている。WindowsNT4.0以前とは名前のつけられ方が微妙に異なっている。

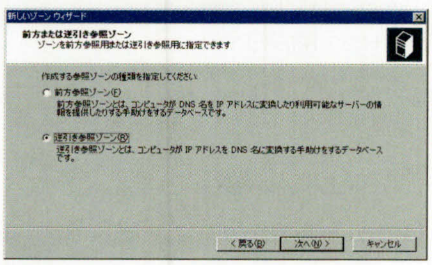


図28

リスト4　sample.co.jp.dns

```
;
;  Database file sample.co.jp.dns for sample.co.jp zone.
;       Zone version:  3
;

@                                                   IN    SOA  server.domain.sample.co.jp.
administrator.domain.sample.co.jp. (
                                    1           ; serial number
                                    900         ; refresh
                                    600         ; retry
                                    86400       ; expire
                                    3600      ) ; minimum TTL

;
;  Zone NS records
;

@                             NS   server.domain.sample.co.jp.

;
;  Zone records
;

server.domain                 A    10.10.1.1
mail                          CNAMEserver.domain.sample.co.jp.
www                           CNAMEserver.domain.sample.co.jp.
```

リスト5　変更後のsample.co.jp.dns

```
;
;  Database file sample.co.jp.dns for sample.co.jp zone.
;       Zone version:  2
;

@                             IN  SOA server.domain.sample.co.jp.  zenji.sample.co.jp. (
                                    2           ; serial number
                                    900         ; refresh
                                    600         ; retry
                                    86400       ; expire
                                    3600      ) ; minimum TTL

;
;  Zone NS records
;

@                             NS   server.domain.sample.co.jp.
                                   NS         ns-tk012.ocn.ad.jp.
                                   MX         10 server.domain.sample.co.jp.
                                   A          200.140.160.178
;
;  Zone records
;

server.domain                 A    200.140.160.178
mail                          CNAMEserver.domain.sample.co.jp.
www                           CNAMEserver.domain.sample.co.jp.
```

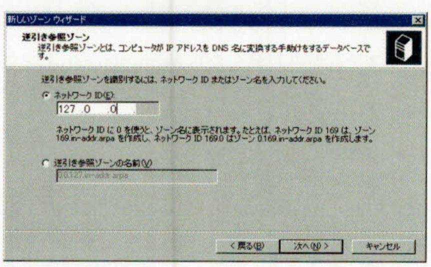


図29

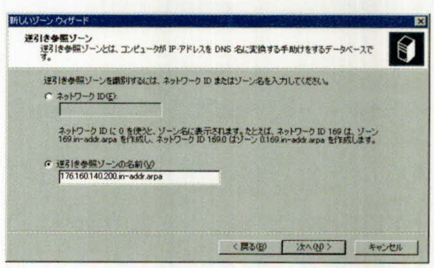


図31

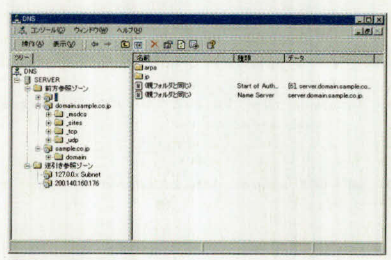


図33

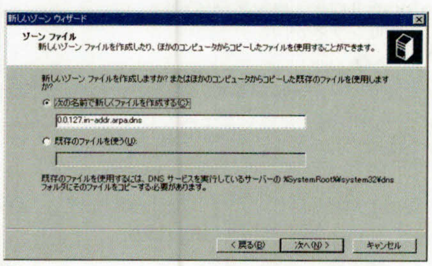


図30

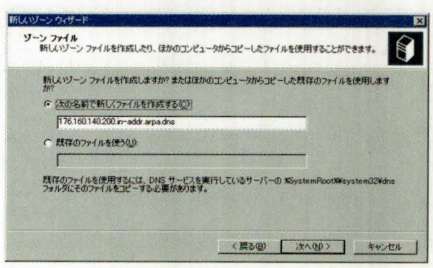


図32

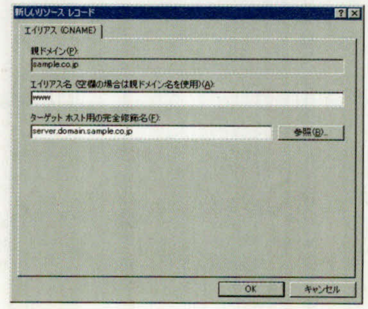


図34

設定ファイルの修正

ここまでの設定を正しく完了するとフォルダ「C:¥WINNT¥SYSTEM32¥DNS」の下に,

sample.co.jp.dns
224.162.139.203.in-addr.arpa.dns
0.0.127.in-addr.arpa.dns

といったファイルが作成されているはずだ。

これをNOTEPADなどのテキストエディタで開いて修正を施していく。

●「sample.co.jp.dns」の修正

変更前がリスト4(sample.co.jp.dns), 変更後がリスト5(変更後のsample.co.jp.dns)だ。上から見ていくことにしよう。

まず, 先頭のほうの「Administrator.domain.sample.co.jp.」という部分だが, これをDNSサーバ管理者のメールアドレスに置き換える。まだ, メールサーバの設定をしていないのでメールアカウントはないのだが, まぁ, これから作る予定のメールアドレスを記入すればいい。ただし, メールアドレスは世間一般で使われている「@」マーク区切りの表記方法とは違う点に注意。たとえば, 今回の例で管理者のメールアドレスを,

zenji@sample.co.jp

とするならば, ここで表記方法は,

zenji.sample.co.jp.

という感じになる。「@」までなく「.」で区切るのと, 最後尾にもう1個「.」がいる点に要注意だ。

中央付近の「Zone NS records」セクションだが, ここにはOCN(NTT)側のセカンダリDNSのサーバ名を追記する。これはNTTから契約後に送られてきた書類に記載されているはずだ(今回の例で示しているものと似たような名前のはず)。ここでは,

ns-tk012.ocn.ad.jp.

を追記することにする。ここでもサーバ名の最後尾に「.」をつけるのを忘れないように。以後も同様だ。

続けてメールサーバの実名を「MXレコード」で指定し, さらに, そのIPアドレスを「Aレコード」でも記述しておく。

MX　10 server.domain.sample.co.jp.
A　200.140.160.178

「server.domain」のIPアドレスが変になっていたらこれも適宜直しておく。

すべての修正を終了したら, 冒頭付近の「serial number」の値を+1して保存終了する。他のDNSサーバはこの値をチェックしてゾーンファイルの更新判断を行っているため, 今後も設定ファイルを修正した場合は保存する前に+1すること。

●「176.160.140.200.in-addr.arpa.dns」の修正

DNS管理者メールアドレスの変更, OCN側のセカンダリDNSサーバ名の追記は先ほどの修正と同じだ。

「Zone records」セクションにはサブネットマスク,

A　255.255.255.240

を追記し, さらにネットワーク名も記述する。

PTR　sample.co.jp.

これ以降は, このネットワークに参加しているホスト名をIPアドレスの情報とともに記述していく。この例ではホストは1台だけなので,

178.160.140.200.in-addr.arpa.

PTR server.domain.sample.co.jp.のみを追記している。IPアドレスは逆順に書き「.in-addr.arpa.」を付加している点に注意。

修正を終了したら, 冒頭の「serial number」の値を+1して保存終了する。

●「0.0.127.in-addr.arpa.dns」の修正

DNS管理者メールアドレスの変更, OCN側のセカンダリDNSサーバ名を追記し, 冒頭の「serial number」の値を+1して保存終了する。

*　　*　　*

3つのファイルの修正が終わったら, サーバをシャットダウンして再起動しよう。

DNSサーバのテスト

システムが再起動したらAdministratorでログインし, DNSサーバが正しく動作しているかテストしてみよう。

MSDOSプロンプトを起動し,

C:¥>nslookup

と入力し, 「nslookup」というDNSサーバの正当性チェックプログラムを起動する。これはWindowsNTをインストールすると最初からインストールされているソフトなのでCD-ROMなどからインストールする必要はない。

プロンプトが,

>■

と出ているので, ここで, さまざまなWebページのURLアドレスなどを入力してみよう。たとえば,

> sony.co.jp

などと入力し,

Non-authoritative answer:
Name:　sony.co.jp
Address:　202.238.80.18

リスト6　176.160.140.200.in-addr.arpa.dns

```
;
;  Database file 176.160.140.200.in-addr.arpa.dns for 176.160.140.200.in-addr.arpa zone.
;      Zone version:  1
;

@  IN  SOA server.domain.sample.co.jp.  administrator.domain.sample.co.jp. (
                                  1            ; serial number
                                  900          ; refresh
                                  600          ; retry
                                  86400        ; expire
                                  3600       ) ; minimum TTL

;
;  Zone NS records
;

@                         NS    server.domain.sample.co.jp.

;
;  Zone records
;
```

リスト7　変更後の176.160.140.200.in-addr.arpa.dns

```
;
;  Database file 176.160.140.200.in-addr.arpa.dns for 176.160.140.200.in-addr.arpa zone.
;      Zone version:  2
;

@  IN  SOA server.domain.sample.co.jp.  zenji.sample.co.jp. (
                                  2            ; serial number
                                  900          ; refresh
                                  600          ; retry
                                  86400        ; expire
                                  3600       ) ; minimum TTL

;
;  Zone NS records
;

@                         NS    server.domain.sample.co.jp.
                                NS          ns-tk012.ocn.ad.jp.

;
;  Zone records
;

                                            A          255.255.255.240
                                            PTR        sample.co.jp.
178.160.140.200.in-addr.arpa.PTR            server.domain.sample.co.jp.
```

という感じの結果が返ってくればOKだ。

さらに，逆に，

>202.238.80.18

とIPアドレスを入力し

Name:　ns.Sony.CO.JP

Address:　202.238.80.18

というふうにホスト名が返ってくれば完璧だ(図34)。

インターネットに接続してみよう

この時点で，サイトの基盤はできあがっている。このサーバマシン上のWebブラウザを起動し，適当なWebページのURLを入力してみよう。めでたくそのページが閲覧できれば，ここまでの設定は正しく行われたことを表している。

場合によっては，Webブラウザを起動するとインターネット接続ウィザードが起動することもある。この場合は「セットアプオプション」で「電話回線またはLANを使って，既存のインターネットアカウントに新しい接続を設定する」を選択(図36)，「インターネット接続の設定」で「LANを使って接続する」を選択(図37)する。以降プロキシサーバの有無やメールアカウント，ニュースグループの設定を聞いてくるがいずれも「いいえ」を選んでおく(サーバパソコン上でメールやニュースを送受したりすることはないため)。

メールサーバを構築する

めでたく自分のOCNエコノミーサイトがインターネットに接続された。今度はメールサーバを自分のサイトにインストールしよう。

●メールサーバソフトはどれにする?

Windows NT4.0/2000 Serverに対応したメールサーバソフトとしては，

・マイクロソフト

「Exchange Server 5.5」(http://www.microsoft.com/japan/backoffice/exchange/)実売価格8万～9万円(エンタープライズ版)

・アスキーネットワークテクノロジー

「Sendmail with POP3」(http://www.ant.co.jp/products/sm25_index.html)実売価格7万円前後

・オープンテクノロジーズ



図34

リスト8　0.0.127.in-addr.arpa.dns

```
;
;  Database file 0.0.127.in-addr.arpa.dns for 0.0.127.in-addr.arpa zone.
;      Zone version:  1
;

@  IN  SOA server.domain.sample.co.jp.  administrator.domain.sample.co.jp. (
                        1            ; serial number
                        900          ; refresh
                        600          ; retry
                        86400        ; expire
                        3600       ) ; minimum TTL


;
;  Zone NS records
;

@  NS    server.domain.sample.co.jp.

;
;  Zone records
;
```

リスト8　変更後の0.0.127.in-addr.arpa.dns

```
;
;  Database file 0.0.127.in-addr.arpa.dns for 0.0.127.in-addr.arpa zone.
;      Zone version:  2
;

@                     IN  SOA server.domain.sample.co.jp.  zenji.sample.co.jp. (
                        2            ; serial number
                        900          ; refresh
                        600          ; retry
                        86400        ; expire
                        3600       ) ; minimum TTL


;
;  Zone NS records
;

@                     NS  server.domain.sample.co.jp.
                          NS        ns-tk012.ocn.ad.jp.

;
;  Zone records
;
```

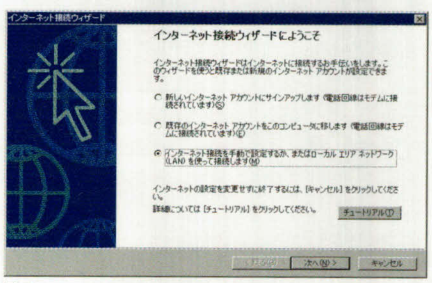


図35

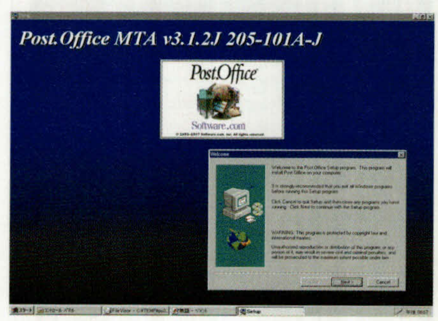


図37

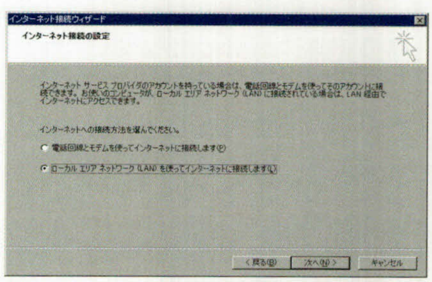


図36

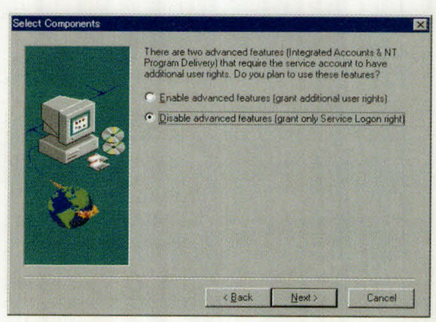


図38

「PostOffice 3.5.3J」(http://product.opentech.co.jp/product/post/index.htm)実売価格4万円前後(50アカウント)
などがある。

どれでもいいのだが，「PostOffice」はうれしいことに，10アカウント対応バージョンが無料で提供されているので，今回はこれを使うことにする。SOHO環境ならば10アカウントもあれば十分だし，個人環境ならばなおさらだ。

PostOfficeのダウンロードはオープンテクノロジーズのWebのダウンロードページ(http://product.opentech.co.jp/download/postoffice.html)から行える。

●PostOffice3.5.3.をインストールしよう

ダウンロードしたZIPファイルを適当なフォルダに展開すると，[disk1]～[disk6]までのサブフォルダが作成される。ここの[disk1]の中にある「setup.exe」を起動し，インストールを開始しよう。

最初に「WindowsNTにAdministratorでログインしていないと正しくインストールできないですよ」という旨の英文が表示されるので，ちゃんとAdministratorでログインしているのであれば[はい]ボタンを押して続行する。

このあと「セットアップ情報の確認要請」「ライセンス内容の承認要請」などのメッセージウィンドウが開くので，読んで[OK]ボタンを押して先に進んでいく(図38)。

●アカウントの統合を行うか

「メールアカウントとWindowsNTドメインのアカウントを統合させるか」という旨の質問をされるが，これはどちらか好きなほうを選べばいい。今回構築しているサイトは特にそういったサービスを必要としないので「Disable」のほうを選んでおく(図39)。

●管理者アカウントの設定

続いてPostOfficeの管理者アカウントをWindowsNTの既存アカウントから選択するのか，それとも新しく管理者アカウントを作成するのかを選択する。ここでは，「既存のアカウントを選ぶ」ということで「Use Existing NT account」を選択しておく(図40)。

この場合，次に出てくるのは「その既存のアカウントはどれにしますか」というダイアログ。この決定権を持つのは自分でいいので，このサーバの「Administrator」アカウント名を入力すればいいだろう。ただし，ドメイン名から記述しなくてはいけないので注意すること。

ドメイン名は「プライマリドメインコントローラの作成」のところで作成したものを入力する。ここでいうドメインはNTドメインのことなので混同しないように。今回の場合は「SAMPLE」として命名したので結局，

SAMPLE＼Administrator

と入力することになる。ドメイン名とアカウント名の間は「＼」(バックスラッシュ)で区切るのを忘れないように。パスワードはそのアカウントのパスワードをそのまま入力する(図41)。

●管理者情報の設定

次は管理者の情報を要求してくるので正しく入力する。シリアル番号は，今回のように「10アカウントの無料版」として使うのであれば「Trial」のままでいい。「Trial」とはいっても制限はアカウント数制限だけで使用期限などはないので安心してほしい(図42)。

今後，アカウントライセンスをなんらかの方法で入手したり，製品版を購入した場合は，もう一度Setup.exeを実行して画面の指示に従えばそれまでの環境を継承したライセンスのアップグレードが行える。

●ドメイン名の入力

次に取得したドメイン名(今度はNTドメインではなくてインターネットのドメイン名)を入力する。このときホスト名やサーバ名を入力してしまわないこと。今回の例では，

sample.co.jp

を入力すればいいことになる(図43)。

このあと確認のダイアログが出るので，設定に間違いがないことを確認したら[OK]ボタンを押す。

●ポート番号の指定

今度は「PostOfficeのWWWサーバ機能のポート番号を入力しろ」といってくる。

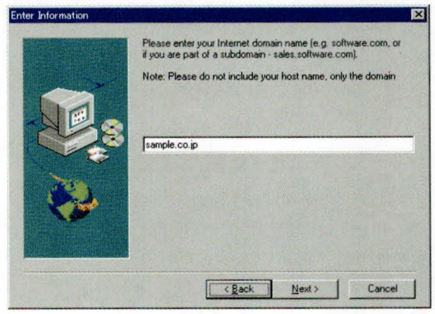


図43

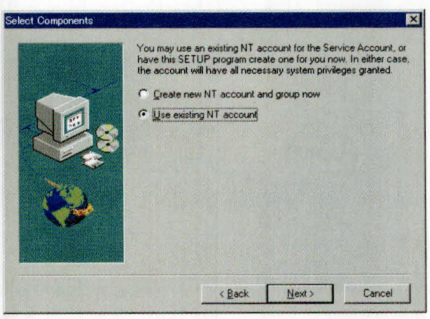


図39

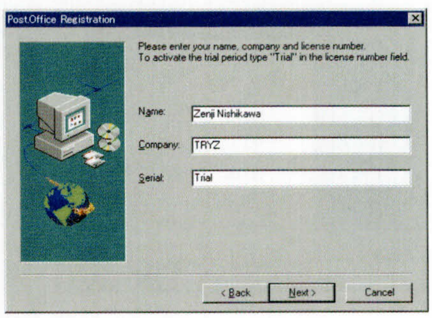


図41

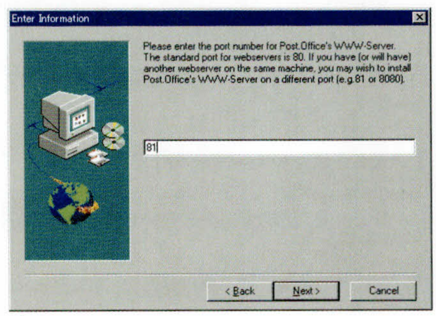


図44

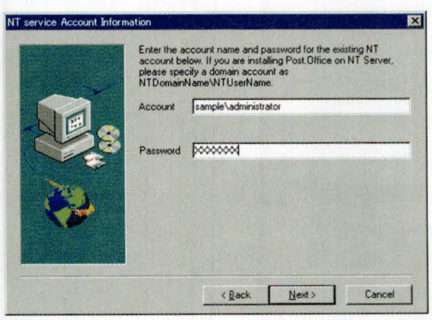


図40

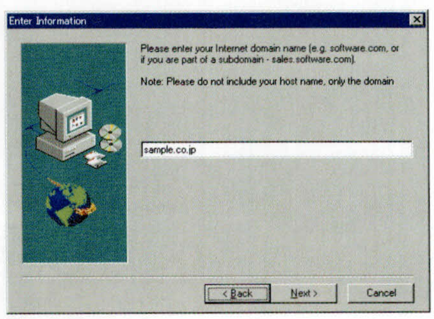


図42

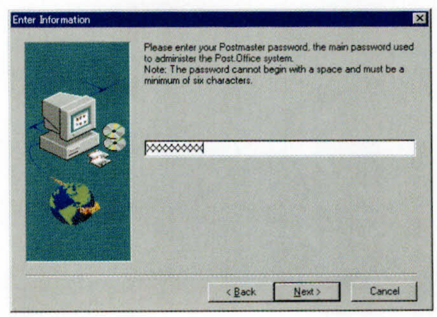


図45

PostOfficeでは，メールアカウントの作成やそのほかのメンテナンス作業をWebブラウザを使って行うことができるのだが，この際PostOfficeがHTMLをWebブラウザに発信する機能を果たす。その際のポート番号をいくつにするか……というのがこの質問だ。

通常HTMLの発信は「ポート番号80」を使うのだが，今回の例ではWWWサーバソフトとしてIISを選択しており，これがポート番号80を使ってしまっているために80以外の値を入力する必要がある。おそらく，デフォルトでは「81」という値がテキストボックスの中に記入されていると思うので，これをそのまま使うことにする。実際セキュリティ上，外部から勝手にメールアカウントを作成されたりしないためにも，このポートは「80」にしないほうがいいだろう（図44）。

●Postmasterのパスワードの設定

最後に「PostOfficeの管理者アカウントである『Postmaster』のパスワードを入力してくれ」という質問が投げかけられる。なにを入力してもいいのだが，忘れると大変なのでこのマシンのAdministratorアカウントと同じパスワードにしておいたほうが無難だ。なお，6文字以上ないとパスワードとして認められないので注意すること（図45）。

ここまでの入力が終わるとインストーラはサーバマシンへのファイルコピーを開始する。

●PostOfficeの起動に失敗した場合

コピー完了後，入力した情報に誤りがあると「PostOfficeの起動に失敗した」という内容のメッセージが表示される。この場合でも，メールサーバソフトのインストール自体は完了している。パニックにならないように「コントロールパネル」-「サービス」を開こう（図46）。

この中から「PostOffice MTA」をダブルクリックしてプロパティを開き，アカウントやパスワードの正当性を確認してから［開始］ボタンを押そう。

メールサーバの設定

インストールが完了したらWebブラウザを起動し，PostOfficeのメニュー画面を出そう。

Webブラウザのアドレス入力欄に，

　サーバ名.ドメイン名:ポート番号

の順で入力する。今回の例ならば，

　server.sample.co.jp:81

という感じになる（図47）。

ここで，管理者のメールアカウントとパスワードを入力し［認証］ボタンを押せばPostOffice管理者のメールアカウントの設定になる（図48）。

管理者といってもいまインストールしている自分のことなので，必要事項をどんどん記入していこう。

ちょっとわかりにくいのは「E-mailアドレス情報」という項目だ。ここは，「どのようなメールアドレス宛に送られたメールを管理者宛と判断するか?」を設定するところだ。たとえば管理者の名前が西川善司（Nishikawa Zenji）だったとすると，第三者が管理者にメールする際，

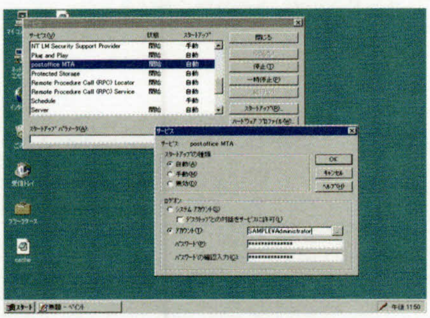
　zenji@sample.co.jp
　zenjin@sample.co.jp
　zenjinishikawa@sample.co.jp
　nishikawazenji@sample.co.jp

という間違いパターンが考えられる。こうした間違ったメールアドレスでも，管理者宛を指しているのはわかりきっているから管理者が実際に受信してしまおう……というのがここの設定項目の意図だ。適当にメールアドレスを列記してもいいし，正しいアドレスをひとつだけ入力するのでもいい。

画面写真ではよく見えていないが，その下には「ローカル配信情報」という設定セクションがある。ここで管理者アカウント名の設定を行わせているので，「POP3配信」チェックボタンをチェックオンし，「POP3ログイン名」を記入する。通常は自分のメールアドレスにする予定のアカウント名を入力すればいいのだが，高いセキュリティを確保するために，あえてメールアドレスとは異なるアカウント名を指定することもできる。

さらにその下には「転送先」という項目のテキストボックスがある。これは「メール配信処理などに問題が発生したとき，そのエラーメッセージなどをどのメールアドレスに配信するか」を設定するものだ。別プロバイダのメールアドレスを持っていれば，それを記入しておけばいいが，特にそうした転送処理が不要ならば空欄でもかまわない。

入力が完了したら［送信］ボタンを押そう。


図46

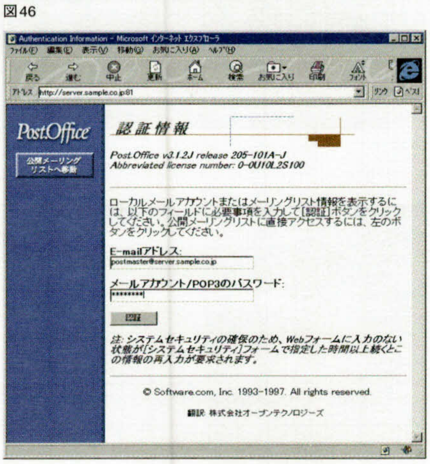


図47

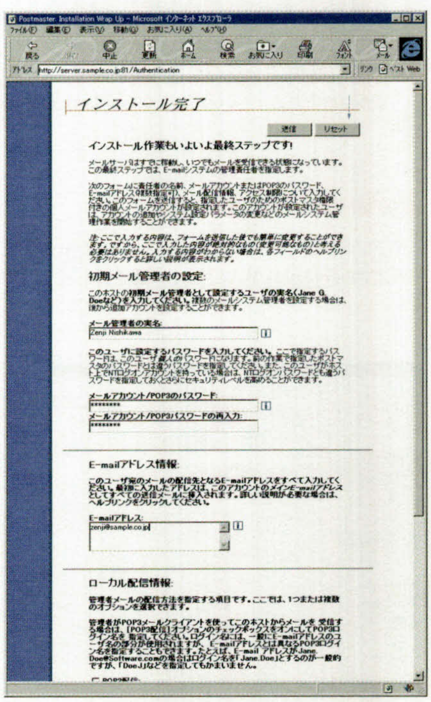


図48

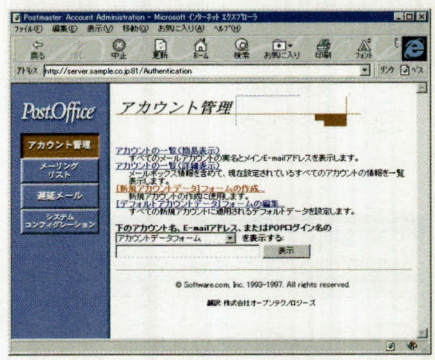


図49

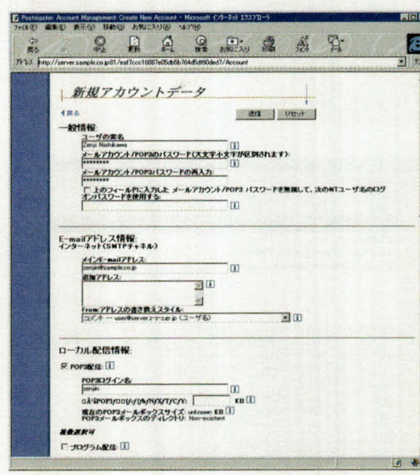


図50

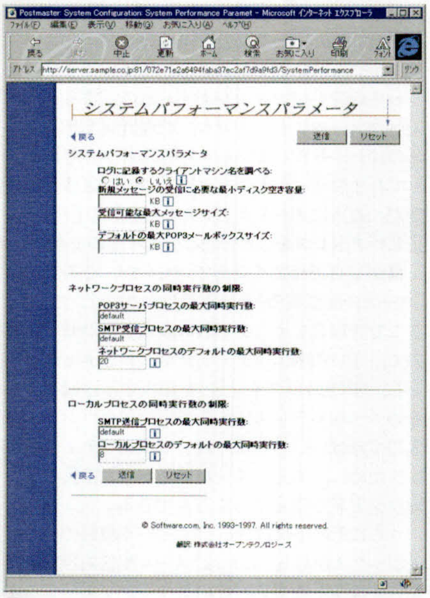


図51

メールアカウントを作ろう

さあ，今度はメールアカウントを実際に作ってみよう。

もう一度Webブラウザを起動し，管理者アカウントで認証する。今度は，図49が現れるはずだ。そしてここで，「アカウント管理」の中の「新規アカウント作成」を選択すると図50のようなアカウント作成画面が出てくる。さまざまなオプションがあるのだが，入力すべき項目はそれほど多くない。ここでは最低限入力するべき項目について解説をしておこう。

●ユーザーの実名

そのまま名前を記載する。同姓同名がいてもおかしくはないので，メールサーバ内で一意である必要はない。

●メールアカウント/POP3のパスワード

パスワードは大文字と小文字が区別される点に注意して入力する。

●メインE-mailアドレス

送ったメールの「From」のところに表示されるメールアドレス。普通は今作成しているメールアカウントのメールアドレスを記入すべきだが，別のメールアドレスを指定することもできる。

●POP3配信，POP3ログイン名

POP3配信のチェックボックスをチェックし，POP3のアカウント名を入力する。アカウント名はメールサーバ内で一意でなければならないため，実名が同姓同名の場合でも変える必要がある。

＊　＊　＊

以上の項目を入力し，［送信］ボタンを押せば，ほかの項目が空欄でもメールアカウントが作成できる。

ところで，PostOfficeでは，メールアカウントごとにさまざまな制限をつけることが可能だ。たとえば「メールボックスの容量制限」や，「送受可能なメールの容量設定」などは，「システムコンフィグレーション」の中の「システムパフォーマンスパラメータの設定」で行える（図51）。

なお，PostOfficeをインストールした直後はそういった容量制限は一切ない状態になっている。サーバのHDD容量が少なくなってきたり，アカウントが増えてきた場合には，こうしたパラメータの変更も必要になるかもしれないので，「そういう設定ができる」ということくらいは覚えておこう。

WWWサーバを構築する

Webページ（ホームページ）をインターネットに配信するのがWWWサーバ機能だ。そして，ファイルをインターネットへ配信したり，逆にファイルをインターネット側からサーバ側に登録したりするのがFTPサーバ機能である。今度はこうしたサーバ機能を，現在設定しているサーバマシンに持たせてみよう。

Windows 200/NT4.0 serverをインストールしたときに，「Internet Information Server（IIS）をインストールするか」という質問に「はい」を答えていれば，特別なソフトをインストールする必要はない。IISにはWWWサーバ機能，FTPサーバ機能の両方が備わっているのだ。

●インターネットサービスマネージャを起動する

［スタート］メニューから［プログラム］-［Microsoft インターネットサーバ（共通）］-［インターネット サービス マネージャ］とたどり，「インターネットサービスマネージャ」を起動しよう。図52が現れるはずだ。

ここで，「WWW」という名前のついている「server」のアイコンをダブルクリックし，「WWWサービスプロパティ」を呼び出す（図52）。

●デフォルトページを設定する

次に「ディレクトリ」タブをクリックすると図53が出てくる。

この画面の下には「デフォルトドキュメントを使用可能にする」というチェックボックスがあるはずだ。これはWebページのURL（アドレス）に，具体的なファイル名を入力しなかった場合にある特定のページを表示させるかどうか決める設定だ。ここは通常はチェックオン（有効）にしておこう。

この下の「デフォルトドキュメント」はその表示させるドキュメントのファイル名を入力する。デフォルトでは「default.htm」になっていると思う

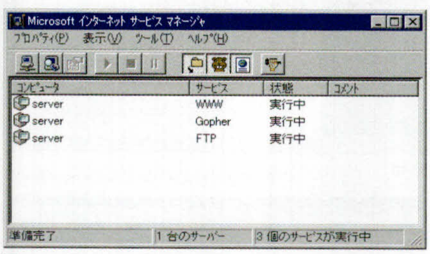


図52

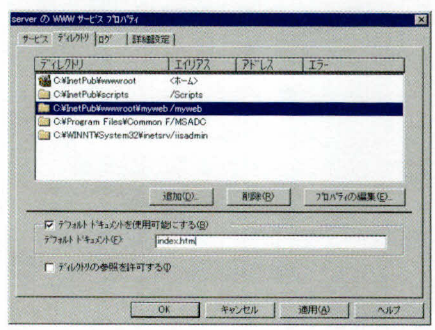


図55

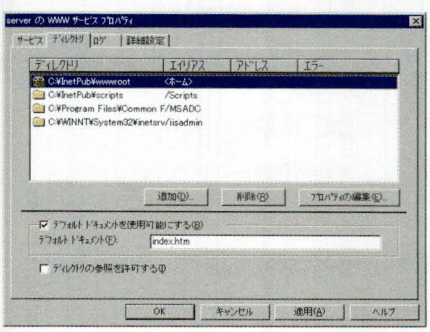


図53

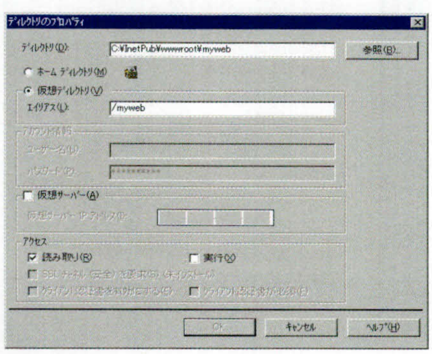


図54

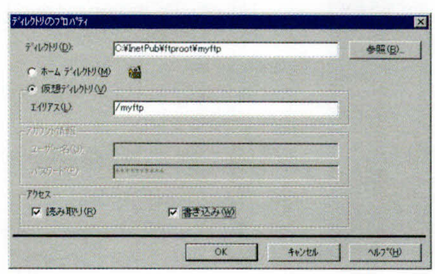


図56

が，慣例的には，

index.htm（あるいはindex.html）

などが望ましいので変更しておこう。

これで，たとえばとある外部ユーザーが自分のWebブラウザに，

http://www.sample.co.jp/myweb/

と，具体的なファイル名を入力せずに，こちらのWWWサーバを呼び出したとしても，

http://www.sample.co.jp/myweb/index.htm

が自動選択されて表示されるようになる。

その下の「ディレクトリの参照を許可する」は絶対にチェックを外しておくこと。ここをオンにしてしまうと，外部ユーザーが，

http://www.sample.co.jp/myweb/

のような感じで，具体的なファイル名を指定せずにこちらのサーバを参照したときに，サーバのディレクトリ構造を表示してしまうからだ。このユーザーにディレクトリ間を移動され，見せたくないファイルなどを勝手に見られてしまう可能性もある。注意しよう。

●Webページを保存しておくフォルダを作成する

ここでは，Webブラウザから，

http://www.sample.co.jp/myweb/

として呼び出せるホームページの設定を行うと仮定する。

まず，Webページのファイルを保存する予定のフォルダを作成しよう。作成する場所は基本的にはどこでもいいのだが，管理のしやすさを考え，すでにできている「C:¥InetPub¥wwwroot」の下に，

myweb

というフォルダを新規作成することにしよう。

いうまでもないことだが自分で作成したWebページのHTMLファイルや各種画像は今後ここに保存していくものとする。

●WebページのフォルダをIISに登録する

画面は依然，インターネットサービスマネージャの「ディレクトリ」タブになっていると思うが，この中の［追加］ボタンを押して「ディレクトリのプロパティ」図54を出す。ここで，先ほど作成したフォルダ（ディレクトリ）をIISに登録するのだ。

画面最上部の項目「ディレクトリ」はどのフォルダを登録するのかを指定するものだ。すでに作成してあるフォルダのパスを入力するか，［参照］ボタンで希望するフォルダを選択しよう。

「仮想ディレクトリ」はインターネット側からどういう名前で参照させるかを決めるものだ。普通は「/（スラッシュ）」をつけて，フォルダの名前を書いておけばいい。

あとの項目はデフォルトでいいのだが，ひとつ覚えておきたいのは下にある「実行」という項目だ。これは，このフォルダ上でプログラムを実行させるかどうかを取り決めるものだ。「プログラム?」と首を傾げてしまう人もいるだろう。普段なにげなく見ているWebページでも，実はプログラムを動かし，その動作により表示の一部を作り出している場合が多いのだ。

もっともわかりやすい例が「アクセスカウンタ」だ。そのページを何人が見てくれたか表示してくれる，アレだ。あれは，ページの参照があると，Webサーバ側のカウンタプログラムが起動され，その演算結果が，ページ内の数値として表示されているのである。もうちょっと高度なプログラム系Webページとしては「掲示板ページ」などが挙げられる。

とにかく，単なるWebページの配信だけでなく，そうしたインタラクティブなWebコンテンツを見せたいというのであれば，この「実行」はチェックオンにしておかなければならない。

●公開されたかどうかテストしてみよう

ここまで入力して［OK］ボタンを押すと図55のようになる。登録したフォルダが一覧に追加されていれば設定完了だ。

Webブラウザを起動し，登録したWebページのURLを入力してちゃんと開けるかどうかテストしてみよう。今回の例でいえば，

http://www.sample.co.jp/myweb/

とURLを入力してそのWebページが表示されれば「成功」ということになる。

ftpサーバを構築する

インターネットサービスマネージャの最初の画面で「FTP」という名前のついている「server」のアイコンをダブルクリックし，「FTPサービスプロパティ」を呼び出す。

あとはftpサーバも，WWWサーバの設定とほぼ同様の手順で設定できる。

ちょっと違うのは「ディレクトリのプロパティ」図56で，ftpの場合は「実行」ではなくて「書き込み」という項目が設けられているのだ。これは外部ユーザーが，自分サーバにファイルの書き込みができるかどうかを設定するものだ。自分のサーバ運用方針に沿い，チェックオン/オフ，どちらに設定しても構わないが，不用意にオンにすると，サーバのディスク容量を悪意あるユーザーに盗用されてしまう可能性もある。よく考えて設定しよう。

クライアントPC設定編

クライアントPCの設定をしよう

サーバ側の設定が終わったので，今度はネットワークに参加させる，クライアントパソコンの設定をしよう。

Windows95/98とWindows NT4.0 workstationの場合に分けて説明するが，いずれのOSも標準的なインストールオプションを選択してインストールがなされており，なおかつInternet Explorer5.0以降（以下IE5.0）がインストールされていることを前提で話を進めることにする。

●Windows95/98パソコンの場合

「コントロールパネル」の「ネットワーク」を開く。「現在のネットワークコンポーネント」の中に「TCP/IP」があるかまず確認して欲しい。

ない場合は［追加］ボタンを押して［プロトコル］→［Microsoft］と項目をたどり［TCP/IP］を選択，これをインストールする。この際Windows95/98のCD-ROMを要求されるのであらかじめ用意しておくこと。インストールが完了すると再起動を要求されるのでこれに従う。

さて，「TCP/IP」がインストールしてあると，「現在のネットワークコンポーネント」の中に，

TCP/IP→○○○○

という表記がいくつか並んでいると思う。このうちネットワークカードの型番とペアになっている

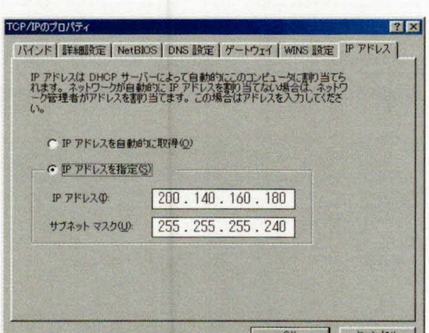


図57

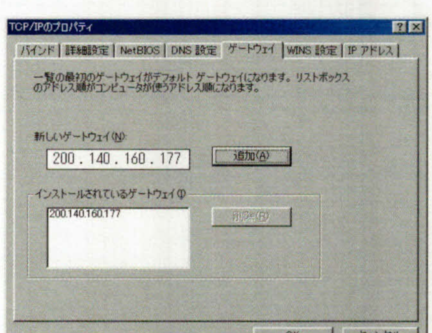


図58

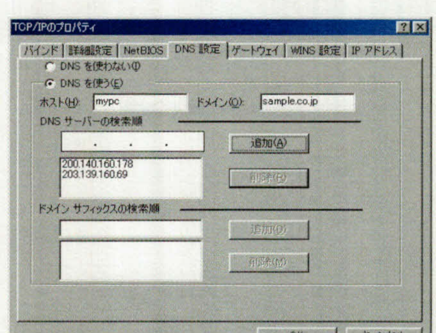


図59

ものをダブルクリックして，その「プロパティ」を開いてほしい。

タブを「IPアドレス」に切り替えここに，そのパソコンのIPアドレスを入力してやる。今回は，

200.140.160.176～200.140.160.191

の16個を取得しているが，先頭3つと，末尾のひとつはシステムにリザーブされているので（176＝ネットワークアドレス，177＝ゲートウェイ，178＝サーバのIPアドレス，191＝ブロードキャストアドレス）結局自由になるのは，

200.140.160.179～200.140.160.190

の12個ということになる。だぶらないように適当に決めて「IPアドレスを指定」を選択，「IPアドレス」「サブネットマスク」の値を入力する。サブネットマスクは，今回の例ではIPアドレスを16個取得しているので，

255.255.255.240

を入力する（図57）。

続いてタブを「ゲートウェイ」に切り替える。「新しいゲートウェイ」の欄に，ルータに割り当てたIPアドレスを入力し［追加］ボタンを押す。今回の例では，

200.140.160.177

だ（図58）。

タブを今度は「DNS設定」に切り替え，「DNSを使う」を選択する（図59）。

「ホスト」には，このパソコンのホスト名を適当に入力し，「ドメイン」には取得したドメイン名を記入する。ややこしいのだがここの「ドメイン」はWindows2000のドメインとは違い，ネットワーク上のコンピュータの集合体のほうのドメインだ。今回の例では「ホスト」には，

mypc

を，そして「ドメイン」には，

sample.co.jp

を入力している。

続いて，DNSサーバのIPアドレスを入力し，［追加］ボタンを押す。今回セットアップしたDNSサーバのIPアドレス，そしてOCN側のセカンダリDNSのIPアドレスも忘れずに入力しよう。OCN側のセカンダリDNSサーバのIPアドレスはNTT側から送られてきた書類のどこかに記載されているはずなので確認しよう。

今回の例では自分のところのプライマリDNSサーバのIPアドレスとして「200.140.160.178」を，セカンダリDNSサーバとして東日本の「203.139.160.69」を入力している。

ここまで入力したら［OK］を押し，設定を終了する。再起動を要求されるのでこれに従う。

●Windows2000 Professional/NT Workstationパソコンの場合

Windows2000 Professional/NT Workstationの場合もWindows95/98とほとんど同じ手順で進めればいい。

「コントロールパネル」の「ネットワーク」を選択，「プロトコル」タブの中の「TCP/IP」をダブルクリックし，このプロパティを開く。

「IPアドレス」タブの中には「IPアドレス」「サブネットマスク」「デフォルトゲートウェイ」の欄があるのでWindows95/98の例のときと同じように必要事項を入力していく。「デフォルトゲートウェイ」とはゲートウェイ，すなわちルータのIPアドレスを入力すればいい（図60）。

「DNS」タブの設定内容もWindows95/98の場合とほぼ同じだ（図61）。

インターネット接続ウィザードを使う

最後に，クライアントPCの最終的な設定を行う。これはWindows95/98，WindowsNT 4.0/2000いずれの場合も共通の方法で行える。

IE5.0をインストールしているならばデスクトップに「インターネットに接続」というアイコンができているはずだ。これを起動しよう。

●インターネット接続ウィザード～インターネット接続環境設定

専用線でしかもLAN経由で接続するので，ウィザードには以下のように答えていく。

・インターネット接続ウィザードにようこそ（図62）

「インターネット接続を手動で設定するか，またはローカルエリアネットワーク（LAN）を使って接続します」を選択する。

・インターネット接続の設定（図63）

「ローカルエリアネットワーク（LAN）を使って接続します」を選択する。

・ローカルエリアネットワークのインターネット設定（図64）

プロキシサーバは使用していないのでチェックをすべて外す。

●インターネット接続ウィザード～メールアカウントの設定

・インターネットメールアカウントの設定（図65）

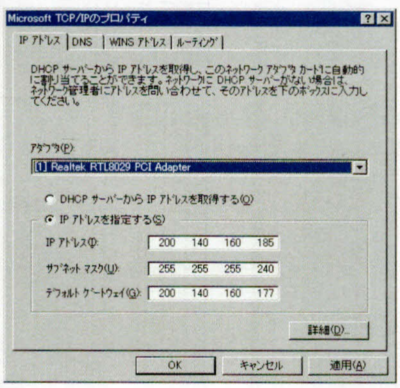


図60

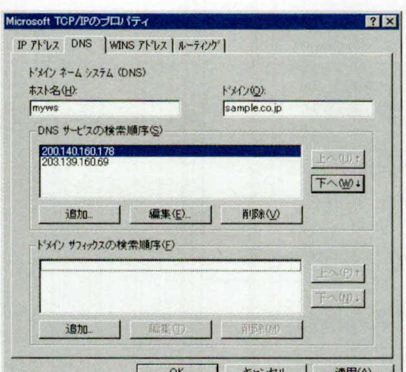


図61

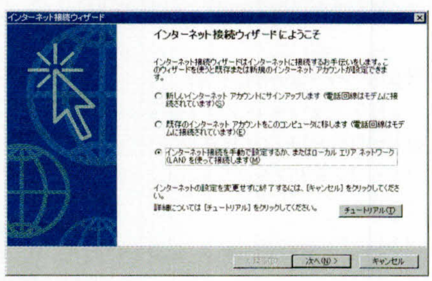


図62

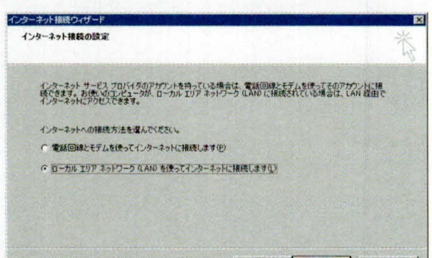


図63

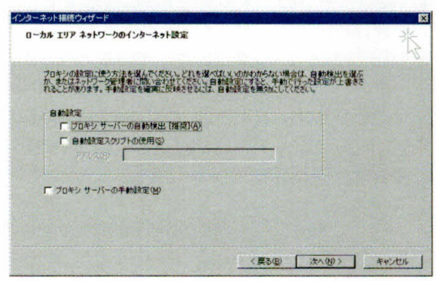


図64

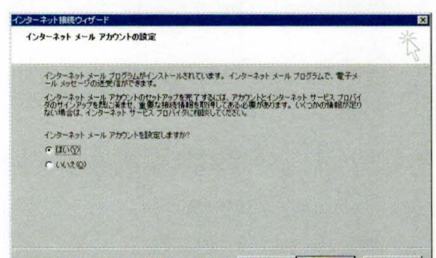


図65

接続ウィザードは続いてメールアカウントの設定へ突入する。

・名前(図66)

送信者の表示名となる名前を入力する。日本語(漢字表記)でもいいが，普通はローマ字表記にする。

・インターネット電子メールアドレス(図67)

自分のメールアドレスを入力する。ここは今回構築したメールサーバに登録したメールアカウントを書いておく。あえて違うメールアドレスを記入することも可能だ。

・電子メールサーバ名(図68)

今回構築したメールサーバ名を入力する。今回は1台のサーバにすべてのサーバ機能を持たせているため，

server.sample.co.jp

とするわけだが，DNSの設定のときに別名「MAIL」を定義しているので，

mail.sample.co.jp

でもいい。

受信，送信ともひとつのサーバで処理しているので，受信メールサーバ，送信メールサーバともに同じサーバ名を記入する。

・インターネットメールログオン(図69)

アカウント名とパスワードを入力する。Post Officeで作ったアカウント名とパスワードを正しく入力する。

・インターネット接続ウィザードを終了する(図70)

これで，すべての設定が完了する。

●OutlookExpress5.0からアカウントを設定する場合

メールアカウントの設定はIE5.0に付属するOutlookExpress5.0(以下OE5.0)からも行える。

OE5.0を起動したらメニュー「ツール」の「アカウント」を選択する。ここで図71のようなウィンドウが出るのでこの中の右上の[追加]ボタンを押し，項目「メール」を選択する。あとは前段で解説した接続ウィザードとまったく同じ手順で設定が行える。

●自分宛にメールを出してみよう

すべての設定が完了したら自分宛にメールを出してみよう。もし届けば，設定はうまくいっていることになる。

ほかのパソコンも同様に設定しよう

これで，このパソコンもインターネットへ接続できるようになった。ほかのパソコンについても同様に設定すればOKだ。ただし，各パソコンに割り当てるIPアドレスはだぶらないように気をつけること。

もし，クライアントの数が増え，IPアドレスが不足してきたら，ルータの「IPマスカレード機能」などを使うといい。これは取得したIPアドレスとLAN側のIPアドレスを自動変換する仕組みのことだ。

もし「同時にインターネットにアクセスするパソコンの数がある程度決まっているのにもかかわらず，パソコンの数だけやたら増えてしまった」という場合は，ルータ側の「DHCP機能」を使うといいだろう。

詳しくはルータのマニュアルを参考にしてほしい。

終わりに

「OCNエコノミーによる常時接続環境を難しい話抜きでとにかく構築できるようにする!」というのをテーマに掲げて書いてきたので，技術的なバックグラウンドの解説はほとんどせずに，設定方法だけを説明してきてしまった。

よって「どうしてこういう設定をすべきなのか」がさっぱりわからなかったかもしれない。そうした疑問点は浮かんだその都度メモ用紙に書き留めておき，あとで念入りに調べて勉強しよう。

OCNエコノミーで「インターネットに常時接続でき，なおかつインターネットに公開できる自分だけのサーバを持てる」ということは無限の可能性を秘めている。しかしその半面「悪意あるユーザーにいつ攻撃を受けるかわからない」という怖い面も持っている。

インターネットの「可能性」を探求したり，「危険性」を理解するためには，そうした「インターネットに関する疑問点」「設定に対する疑問点」を正しく理解することが不可欠だ。

ぜひとも，おっくうがらずに，その疑問点1つひとつをちゃんと調べて理解していこう。

え? なにで調べたらいいかわからないって?

せっかく常時接続環境が整ったんだからインターネットで調べなさいな。

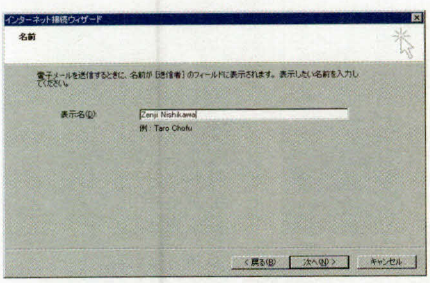


図66

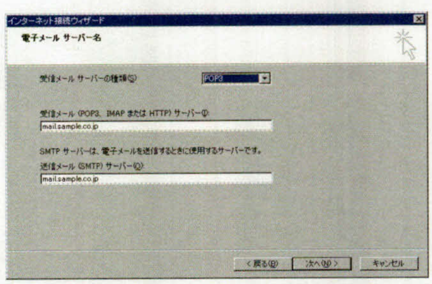


図68

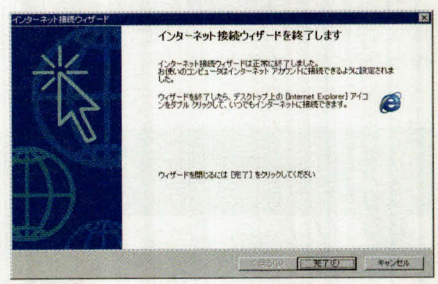


図70

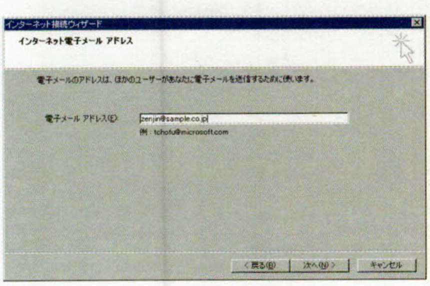


図67

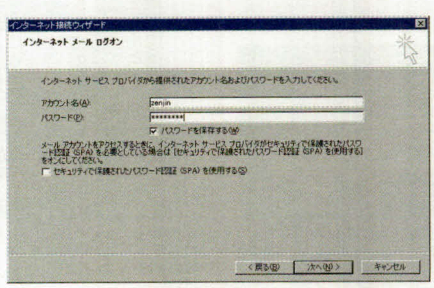


図69

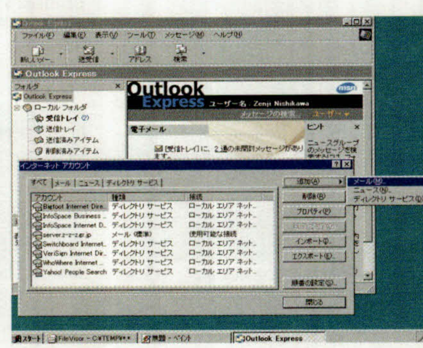


図71

FlexATXによるオリジナルPCの組み立て

X Coppermine DCLXVI-DVD登場

菊地　功　Kikuchi Isawo

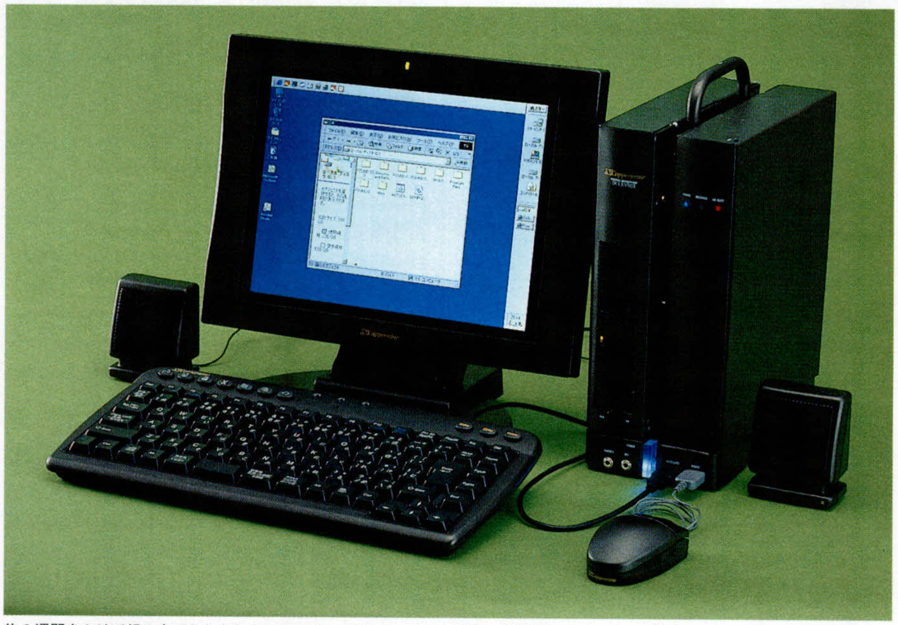
これまでのあらすじ

復刊号で発表された初号機，X Celeron CDXII-CDはそれなりに大きな反響を得た。ラオックス・ザ・コン館に展示されたりもしたのだが，材質がダンボールということで，世間ではどうもただの冗談企画だと受け取られているフシがある（冗談企画だけど）。

しかし，あれはあれで強度的な配慮もされており，それなりに細かく作り込まれている。実際に菊地君宅でマシンはちゃんと実用されているし，機能的には電源があって，ドライブなどの固定スペースがあるだけというそこらのPCとなんら変わりはない。

世間にあるような箱だけの形のPCでは満足できるものはない。デザインPCと呼ばれるものも現れてきているが，液晶ディスプレイのオマケみたいなデザインだったり，丸けりゃいいのか？とかどうもいまいちだ。主張を持った美しいデザインという面ではいまだにこの形を超えるものは存在しないのではないだろうか。

ということで，より完成度を上げた実戦配備仕様の弐号機の組み立て（この程度では自作とか製作とは呼ばない）が始まった。

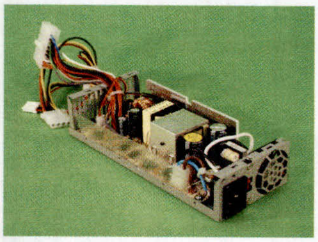


約3週間をかけて組み立てられたX Coppermine DCLXVI-DVDの雄姿。液晶ディスプレイ，光学式マウス，マルチメディアキーボード（邪魔なキーがあるだけだが）との組み合わせだ





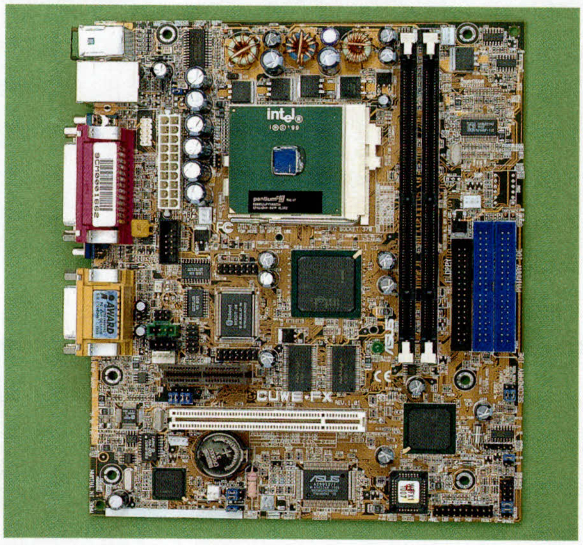
CD-ROMドライブにはノート用の薄型ドライブを使用する。最近は割といろんなところで入手可能だ



今回のキーポイントとなるNLX電源。NLX電源にもいろいろなサイズがあるが（規格はない），幅が狭くて（43mm）細長い，いわゆる羊羹電源（というかは定かではないが）を利用する



ノート用ドライブはそのままでは接続できないが，変換コネクタを利用することで，普通のIDEフラットケーブルを接続できる



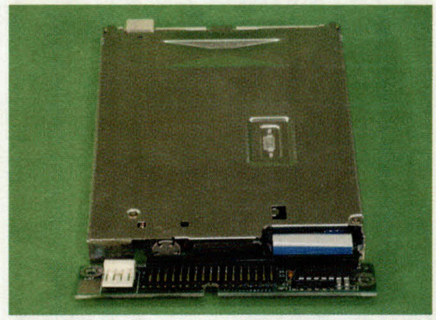
これが噂のFlexATXマザー，ASUSTeKのCUWE-FX。チップセットにi810Eを搭載し，オーディオコーデック，100BASE/TXまでを装備する（i810モデル，LANなしモデルもあり）。microATXと比べてもさらにひと回り小さい。ソケットに載っているのはCoppermineことPentiumIII/500EMHz



ドライブ自体はノート用のものが流用されているようで，ネジを外すだけで簡単に分離できる

FDDもノート用の薄型を入手することは可能だが，こちらの変換コネクタは一般には流通していない。そこで，FDDとPCカードスロットが一体になったコンボドライブからFDD部分だけを流用することにした

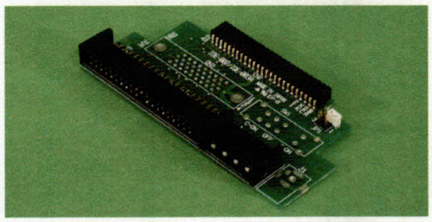




この後ろの基板がミソ。これのためだけに随分高い買い物である。余ったPCカードスロットは，別のマシンにでも使おう

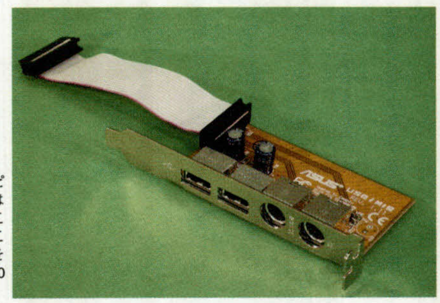


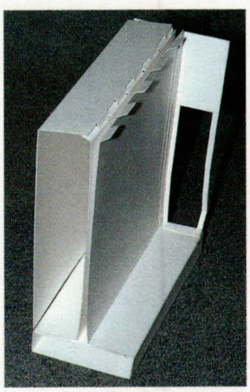
今回利用する2.5インチハードディスク。東芝製8GB，8.4mmハイトの薄型タイプ



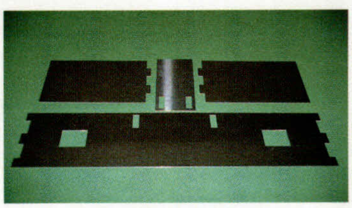
CUWE-FXは背面に2ポートのUSBを持つが，基板上にもUSBのピンを備える（i810EはUSBを4ポートサポート）。このコネクタはASUSTeKのATマザー用のものだが，ピンアサインが同じなので，このマザーでも流用できる。ちなみに右の2つはPS/2とIRだが，ここは使用しない（CUWE-FXのマザー上にはそのピンはない）ので基板をノコギリで切断し，USB部分だけ取り出す。接続ケーブルは別途10ピンのものを用意する

2.5インチのHDDを，通常のIDEフラットケーブルに接続するアダプタ。この手のアダプタはいろいろなものが出回っているが，できるだけ小さいものが望ましい



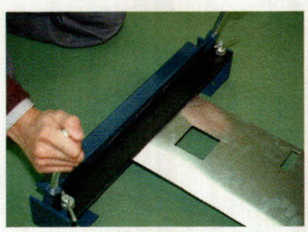


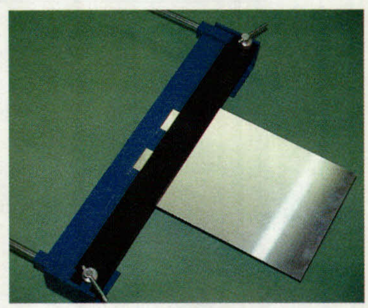
紙で作られた最初の実物大模型。精度を上げるため今回はダンボールではなくてスチレンペーパーを使用している。このあとさらに若干のサイズ見直しを行った2号模型を経て，アルミ板の図面作成が行われた



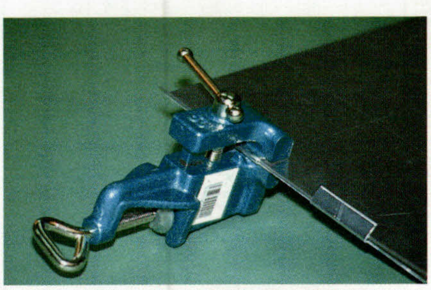
これがボディとなる厚さ1mmのアルミ板。下が低部およびタワー内側となるパーツ，上中は底部の中板，上左右はタワー外側になる扉パーツ。切断・穴あけはお近くの東急ハンズでどうぞ

板の曲げは「どこの家庭の作業台にでも据えつけてあるような」板金折り曲げ機にて。安易な構造の割には，結構綺麗に折り曲げられる。写真は業界では有名らしいホーザンの板金折り曲げ機だ（8,000円程度）

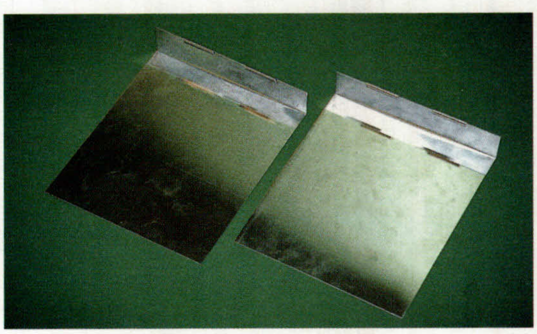




扉の端とボディの上端の矩形歯部分は折り曲げ機で曲げる



さらにピアノ線を通してバイスで挟み，板端を蝶番状に加工する

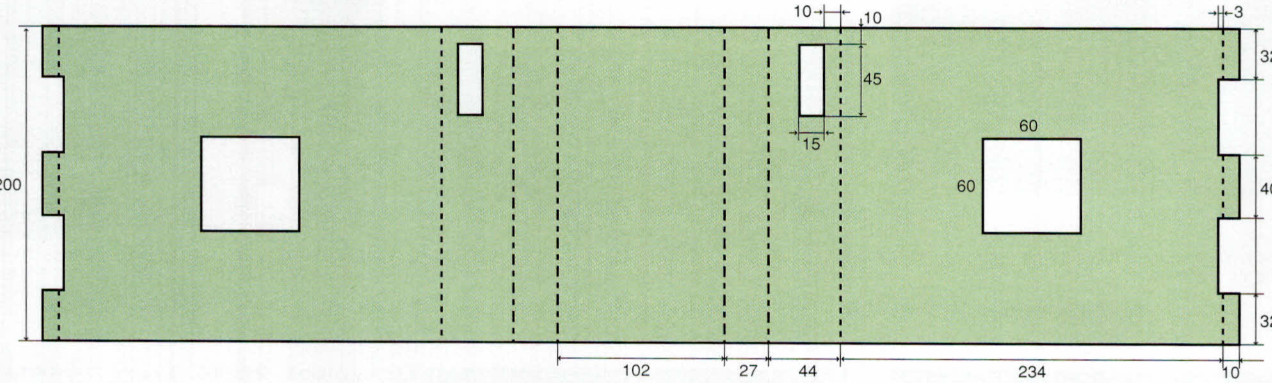


曲げ，蝶番加工が済んだ扉

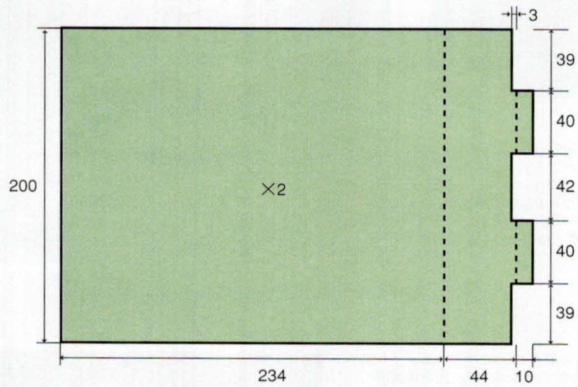


扉とボディの蝶番をあわせてみる。扉は左右に跳ね上げるガルウィング方式を採用

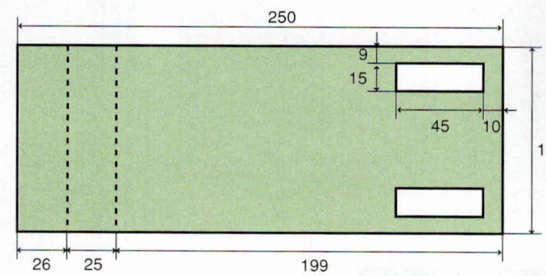
ボディ　板厚1mm



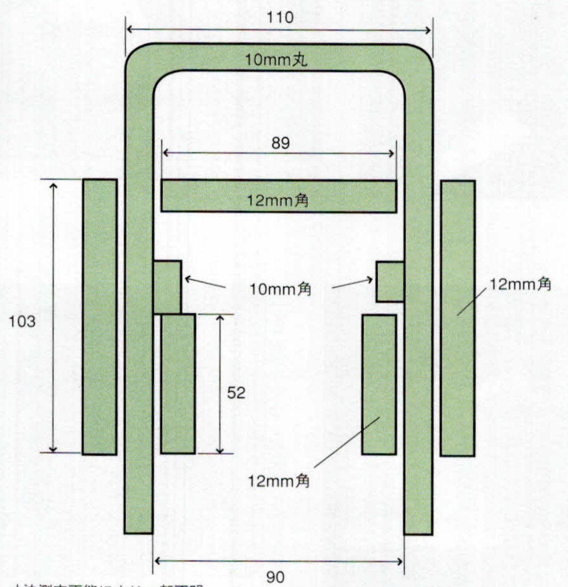
扉　板厚1mm



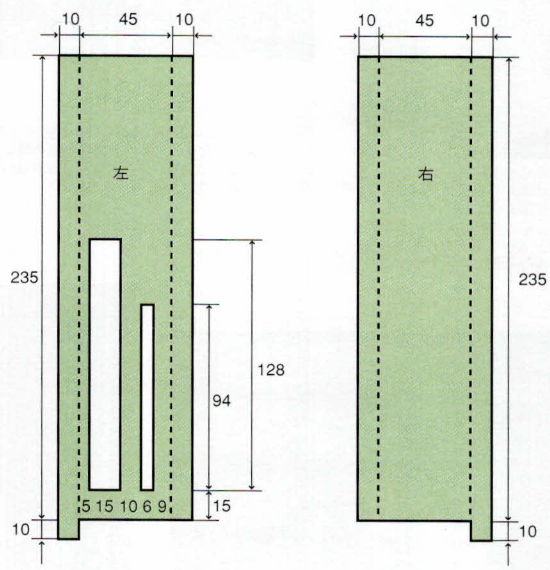
中板　板厚1mm



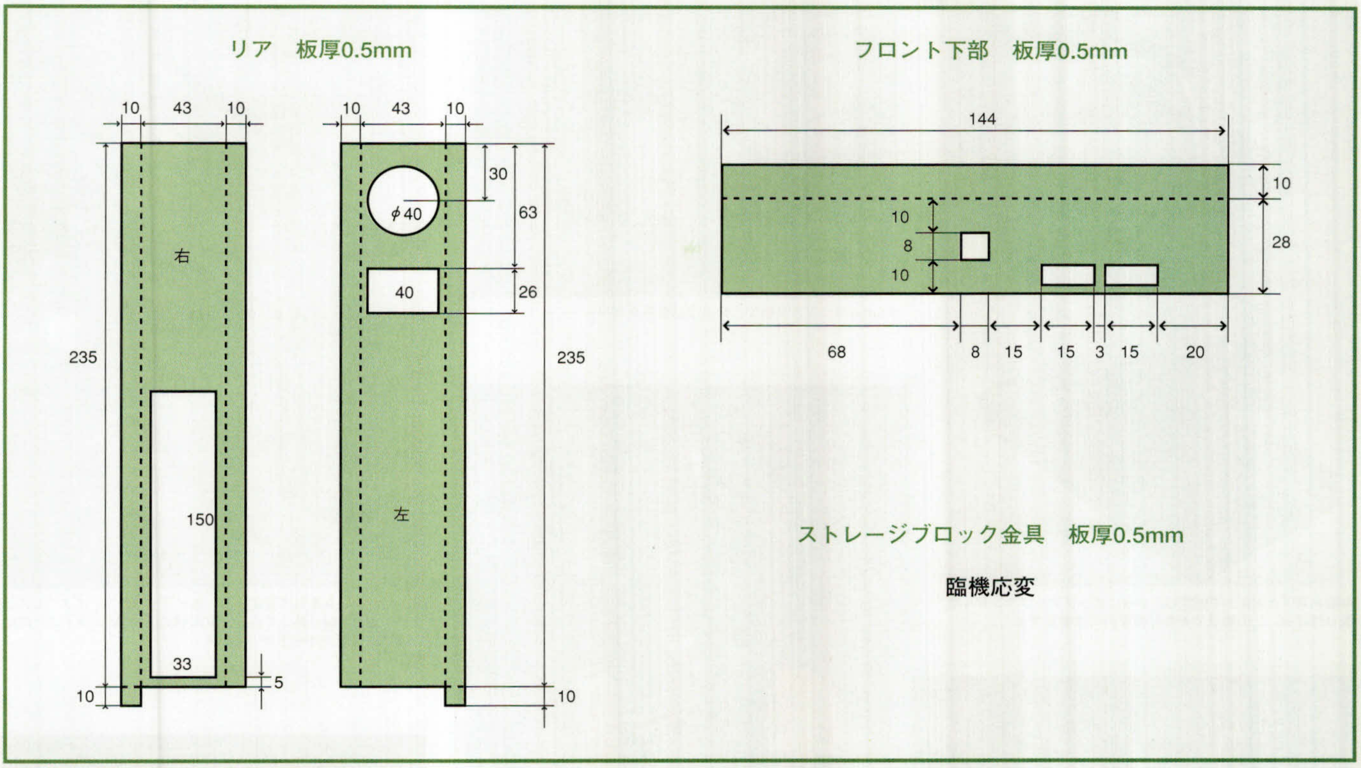
ポップアップハンドル機構



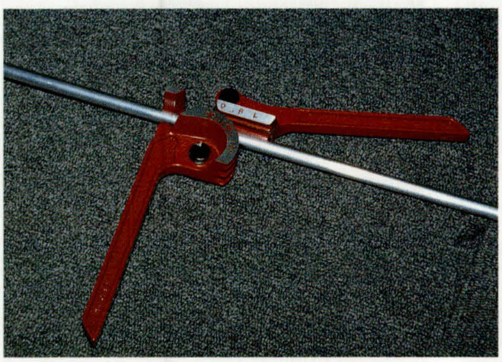
フロント　板厚0.5mm



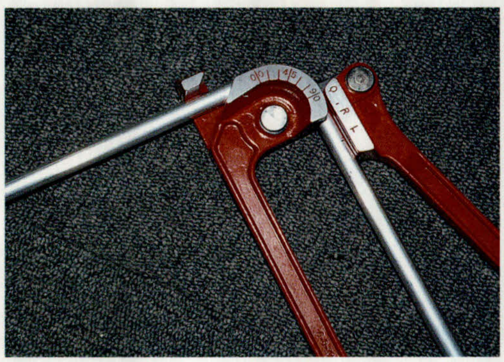
寸法測定不能により一部不明



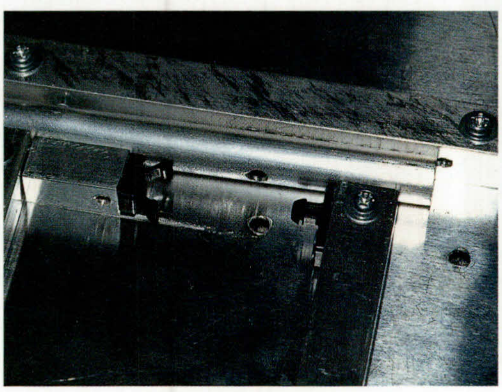
設計図（一部図面紛失のため，実機からの寸法測定）



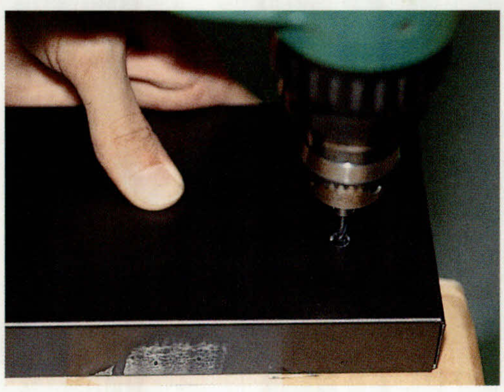
ポップアップハンドルは「どこの家庭の工具箱の中にでも入っているような」パイプ曲げ機で，10mm径のアルミパイプを曲げて作る



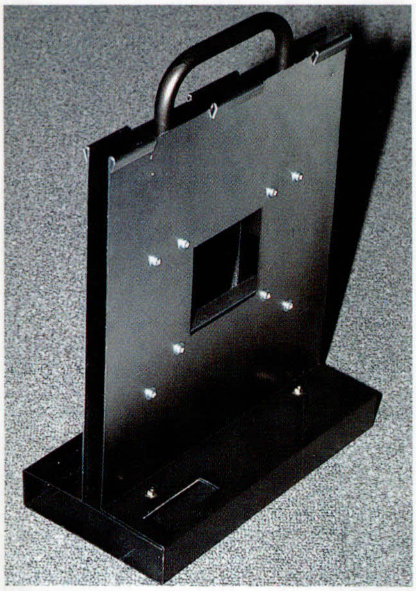
これと「力自慢の工場のあんちゃん」さえあれば，パイプの曲げなんてスイスイ。難点は，コの字に曲げるパイプの幅が実際に曲げてみないとよくわからないということくらいだ



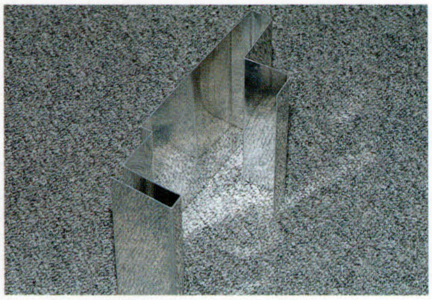
これがハンドルのポップアップ機構。左右の角材の中には引きばねを仕込んである。これだけではハンドルは出っ放しだが，ラッチを仕込むことで，「押すと飛び出る」ギミックを再現



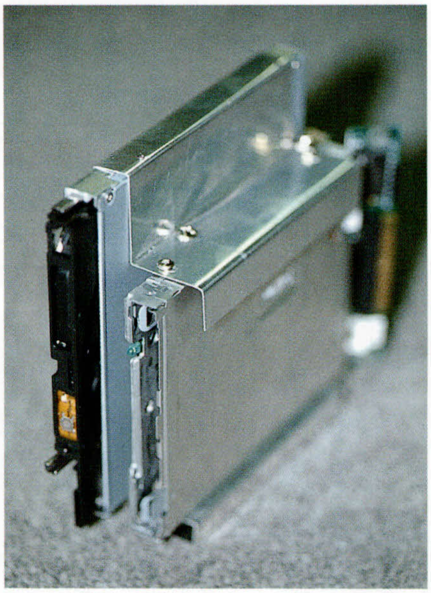
底からドライバーを通す穴を「どこの家庭の倉庫にでも転がっているような」ドリルで開ける



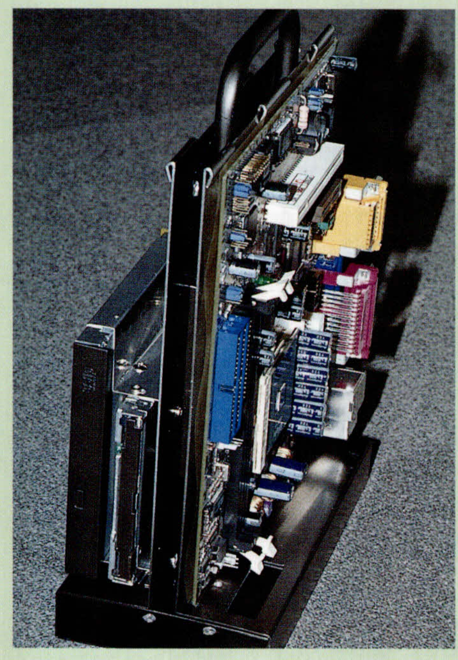
中板をネジとボルトで固定し，さらにポップアップハンドルを取り付ける。この時点でかなり強度的には安定する

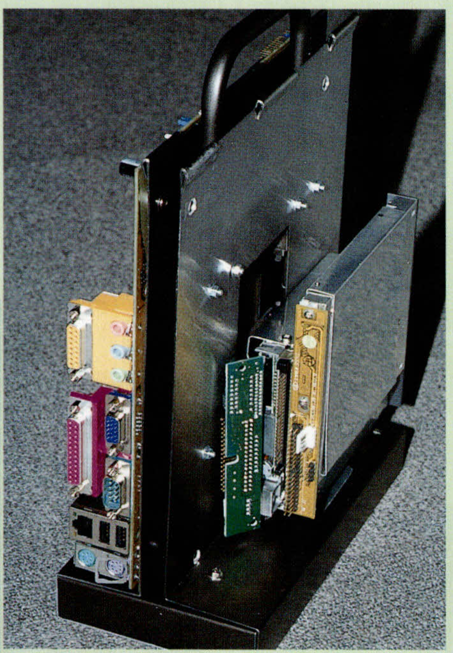


0.5mm厚のアルミ板でこのような金具を作り……

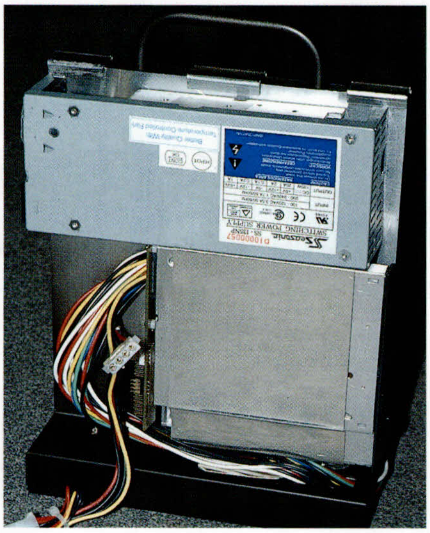


ストレージ類を束ねて固定する。各ドライブのフェイスベゼルは塗装のため，取り外してある。HDDはCD-ROMドライブとFDDの間に挟む形でぶら下がっている

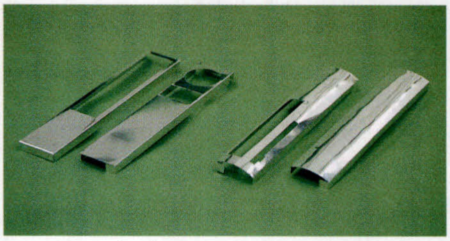




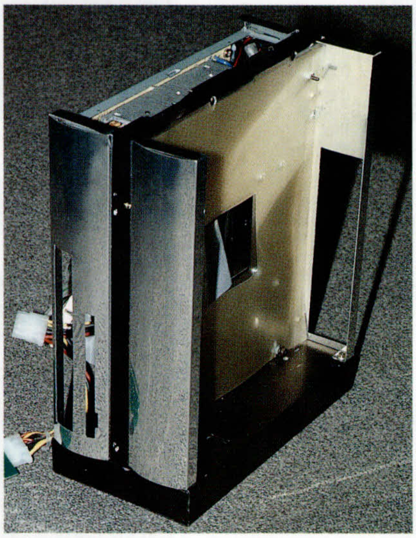
とりあえずストレージブロックとマザーを載せてみた。まだまだこの時点では完成のイメージにはほど遠い



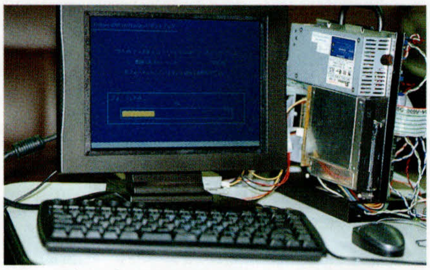
電源はちょうどストレージブロックの上に載っかる形になる。っていうか，本当は少し空くように設計したはずなんだけど



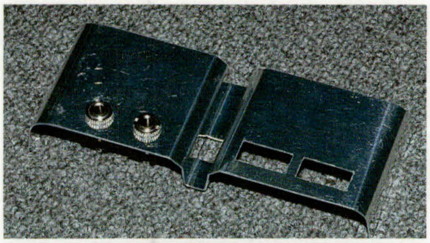
0.5mm厚のアルミで作ったフロントとリアのマスク。フロントは大きなRをつけて膨らませてある

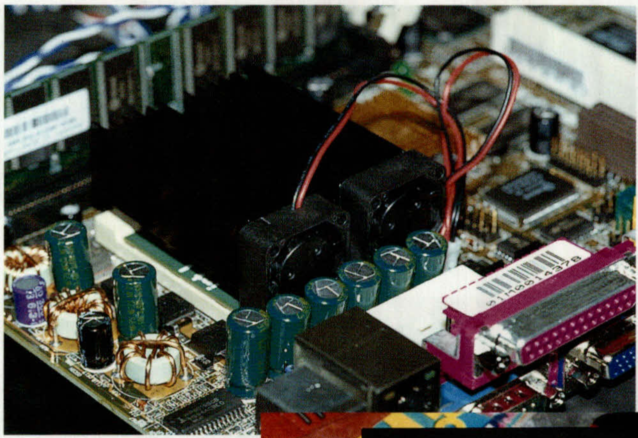


ボディにあわせてみる。だんだんと完成形が見えてきた

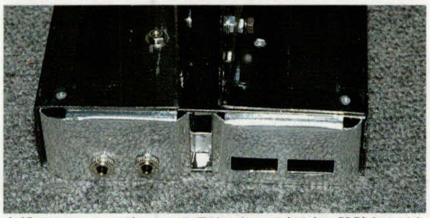


塗装したディスプレイと組み合わせてみる。寸法的にはなかなかいい感じだ。とりあえずここまでで動作チェック。一抹の不安はあるものの，正常に動作しているようだ

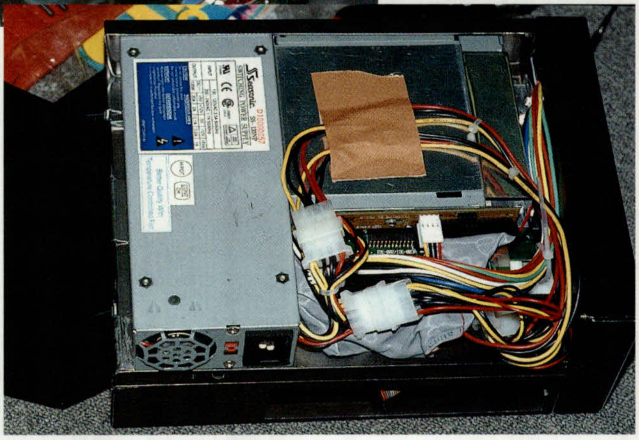




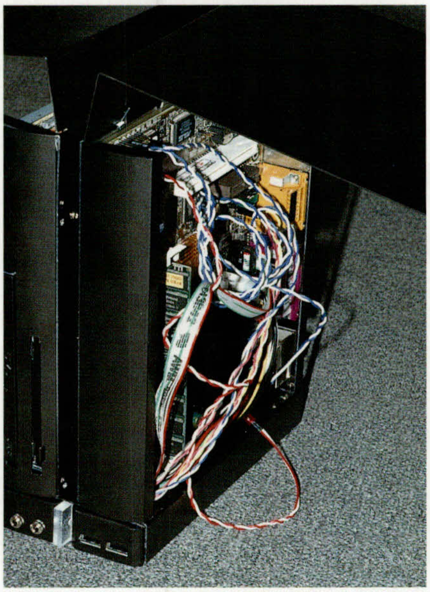
CPUは標準のファンを外して，2cmファンをデュアルで装着。500MHz時にはファンレスで動作検証しており，666MHz時にもこのファンでなんとか動いている。場合によっては今後の強化対象となる部分だ



底部のフロントベゼル。この辺りになってくると，設計なんてあってなきが如し，適当に切って臨機応変なのだが，ぶっつけ本番のわりにはうまくいったような……。ボディに装着。なんかこう……結構イケてない？



いよいよ本格的に組み込み。左タワーから真ん中の穴を通して右のタワーへIDE/FDDケーブルを引っ張り，接続する。左側のタワー内部はこんな感じ。余裕があるようでほとんどない。なんで電源ケーブルってこう太いんだ！

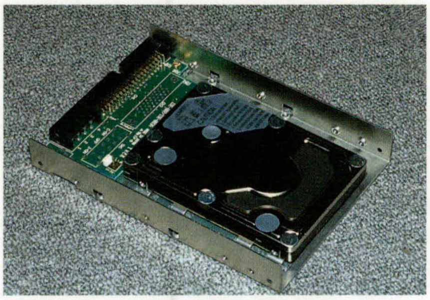


接続は完了。もしゃもしゃのコード類が結構すごいことになっているが，こんなものは，黙って押し込む



小型，低価格がウリの11.3インチSVGA液晶ディスプレイ。結構有名な品かも。TFT液晶ながら，視野角が狭く，階調性が乏しいのが特徴。なんでまっとうな液晶ディスプレイはみんなでかいのばかりなんだ





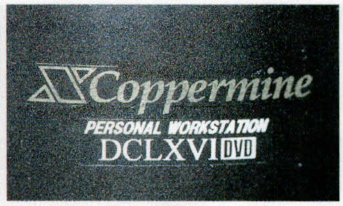
底部がかなり高めの設計だったのは，実は底部を3.5インチシャドウベイにすべしという（U）氏の強い要望があったから。にもかかわらず，もろもろの誤差により3.5インチHDDは入らず。ということで，2.5インチHDDにスペーサーをかませて搭載することに。このサイズでなんと2ドライブ仕様だ。HDDは日立製10mmハイトの10GBドライブ

とりあえずバラして色を塗る。サーフェイサーを塗り，塗料はグンゼ産業のMr.COLOR28番「黒鉄色」で，オーバーコートでつや消しのクリアラッカーを塗っている



最後の仕上げ。アルプス電気のマイクロドライプリンタで打ち出したデカールを貼っていく







いままでのアバウトな作業とはうって変わって細かい作業。斜めにならないように，ゆがまないように，これはこれでなかなか神経を使う

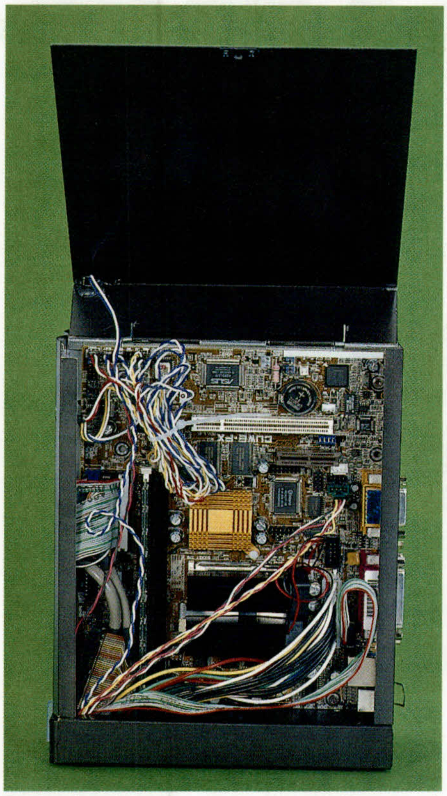


もちろん周辺機器にもロゴは欠かせない

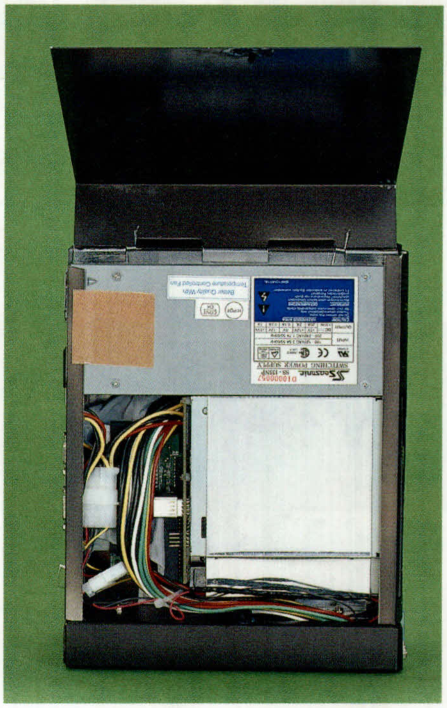




ようやく完成。長い道のりだった。結構いい仕上がりで，編集部内での評判もなかなか。電源スイッチに仕込んだ青色LEDもお洒落。おや？　いつのまにかCD-ROMドライブがDVD-ROMドライブ（DVD-ROMドライブ周りの隙間は，ちょっとした寸法ミス）になっているぞ？　原材料費，東急ハンズでの加工費をあわせると，筐体だけ（電源含まず）でだいたい15,000円といったところか



電源ケーブルの自作などで，ややすっきりした完成版のタワー内部。右扉上部にはリセットスイッチが配置されている



大雑把な原価は表1のとおり。中身自体はほぼハイエンド仕様にしてあるので，価格もそれなりになっている（RAMはさらに増設される可能性あり）。HDDは東芝から8.4mmHDDの大容量版が出たら1台で済むのだが，現状では実用上2ドライブ構成となっている。Celeron/400MHz，PC/100 128MB，HDD6GB，CD-ROMといったローエンド寄りにすれば，60,000円近く安く上がるだろう。なお，10GBドライブは某所から購入した放出品。HDDの固定にはネジとフレーム間にソルボセイン（衝撃吸収樹脂）を挟み込み，本体のゴム脚部分もソルボセイン製のものを使って衝撃を最小限に抑えている。

実機で使われているNLX電源はNLX筐体から抜いてきたものであり，本来のコストはもっとかかっているはずだが（とはいえ，/Vmag編集部に転がっていたものなので実際のコストは発生してないけど），それ以上に単体での入手性を危ぶまれていたものだ。幸い，ぷらっとホームでほぼ同等品が単体発売されていたので，この筐体と同等品を作ることも非現実的ではなくなった。工具さえ揃えれば決して不可能ではないのだ。とはいえ，材料を揃えてさらに図面があっても組み立て手順を熟考しないと組み上がらないと思われるので同等品を作りたい人は注意。

表1 大雑把な原価

CPU	23,000
マザーボード	19,800
メモリ PC133 256MB	33,000
HDD 日立10GB	20,000
HDD 東芝8GB	28,000
アルミ板および加工費	15,000
DVD-ROMドライブ	18,800
FDD	11,800
電源	7,800
コネクタ/ケーブル類	15,000
その他（ネジ，スイッチなど）	5,000
塗料類	6,000
キーボード	4,800
マウス	5,200
液晶ディスプレイ	49,800

●特殊パーツの入手先

CUWE-FX
クレバリー1号店 19,800円
スリムDVD-ROMドライブ/
スリムCD-ROMドライブ変換コネクタ
ツートップ1号店3F 18,800円/1,280円
USB/MIRコネクタ
ツートップ1号店5F 1,280円
USB
NLX電源　ぷらっとホーム3F 7,800円

●使用した工具

ペンチ
ラジオペンチ
ドライバー（＋）
精密ドライバー（＋，－）
板金折り曲げ機
パイプ折り曲げ機
電動ドリル
レーザーソー
ピンバイス
万力
ハンダごて
ハサミ
カッターナイフ
センターポンチ
木槌
ゴムハンマー
ヤスリ
金バサミ

X coppermine DCLXVI-DVD仕様

CPU	PentiumIII/500EMHZ（666MHz動作）
メモリ	PC133 DIMM 256MB
HDD	8GB＋10GB
M/B	ASUSTeK CUWE-FX（Intel810E）
ビデオ	チップセットに内蔵
サウンド	チップセットに内蔵
LAN	10BASE T/100BASE TX
サイズ	270×208×106mm
重量	3.3kg

COLUMN

弐号機のDCLXVIって？

理解していない人も多いと思うので補足説明しておく。型番についている数字は，もちろんCPUクロックを示すものだ。今回は133.3MHz×5で666MHz動作なので，これをローマ数字に直して「DCLXVI」とした。

ローマ数字は，

I＝1
V＝5
X＝10
L＝50
C＝100
D＝500
M＝1000

の7文字だけで構成されている。インド人がとっくに0を見つけて高等数学を展開してたり，無量大数の桁まで数を数えていた時代に平和なことにこれだけで済ませていたわけだ。

基本的に桁の多いほうから並べて各数字を足し算していくが，順番を逆にすると引き算となる。

IV＝4

とかいうのは有名だろう。初号機のCDXIIは，

－100＋500＋10＋1＋1＝412

ということで，Celeron/412MHzとなる。

今回は素直に各数字を並べていくだけなのでDCLXVIということになる。実は700MHzでも安定動作していたのだが，ロゴの見栄えがする型番ということで666MHz動作という仕様に決定した。667MHzだとBIOSはいい張っているが，133.3×5＝666.5ということで，端数は切り捨てるのが潔い。

来年は，X Willamette MMあたりか？

初めて読むMIPS

第2回　MMUと特権命令，およびFPUと命令セットの拡張

中森 章　Nakamori Akira

ゲーム機からワークステーションまで幅広く使われているMIPS Rシリーズのの概要を見ていこう。今回はFPUやMMUといった部分と特権命令，さらに新しく追加されたMIPS16命令について解説する。

はじめに

前回はMIPSの命令セットについて解説した。今回はMMUと例外/割り込み処理，およびFPUに関して解説したい。そのあとで，新しい命令セットについて簡単に解説する。

ところで，FPU以外はMPU特有の仕様が多いので，一般論を述べることは難しい。そこで今回はR4300の仕様を基にR4000などの仕様を適当に取り込んで解説している。特定のMPUに特化しているわけでもなく，かといって，アーキテクチャの一般論をきっちりと押さえているわけでもなく，中途半端といわれたらそのとおりである。まあ，実際にプログラムを組む向きは，対象となるMPUのユーザーズマニュアルで仕様を再確認してほしい。

コプロセッサ0

MIPSアーキテクチャでは例外処理やメモリ管理などの特権機能をCPUとは別のコプロセッサ0というユニットで行うようになっている。コプロセッサ0は特権レジスタ群とMMU（TLB）から構成される（図1）。特権レジスタ（CP0レジスタ）へのアクセスは，CPUとのレジスタ間と，専用命令，

```
mfc0  (Move from Coprocessor 0)
mtc0  (Move to Coprocessor 0)
dmfc0 (Doubleword Move from Coprocessor 0)
dmtc0 (Doubleword Move to Coprocessor 0)
```

などで行われる。MPUを正常に動作させるためには，リセット後に適当な値をCP0レジスタに設定する必要がある。

メモリ管理

CP0の役割のひとつが仮想記憶によるメモリ管理機能である。内蔵されているTLBにより，プログラムで参照する仮想アドレスを物理アドレスに変換する。もし，与えられた仮想アドレスがTLB内になかったり（TLB不一致），書き込み保護違反（TLB変更）の場合は例外が発生する。TLB不一致とは，TLBのタグ部に一致する仮想アドレスが存在しない場合（TLBミス）と仮想アドレスが存在するが無効な場合（TLB無効）の総称である。通常のMPUではTLBミスが発生すると自動的にメモリ内のテーブルサーチを行い，新たなPTE（Page Table Entry）をリードしてきて，TLBを更新するが，MIPS系のMPU（というかRISC系のMPUの多く）では単に例外を発生するだけである。MPU内にTLBのエントリ入れ替えのための複雑な制御回路を持たなくとも，例外処理ハンドラでのソフトウェア処理でも十分高速に実行できるいう目論見があるためである。TLB無効とTLB変更に関しては保護違反なので，通常のMPUでも例外になる。例外処理の性格が異なるため，TLBミスとそれ以外では例外ベクタ（例外発生時の分岐先）も異なっている。

（1）TLBの構成

R4000以降，MIPS系のMPUは64ビットプロセッサであり，アドレス空間に関して32ビットモードと64ビットモードを持っている。TLBの各エントリも32ビットモードと64ビットモードで若干異なる。図2にTLBエントリの形式を示す。各エントリはCP0レジスタである，エントリHi，エントリLo0，エントリLo1，ページマスクレジスタに対応する領域を持っている。TLBは，32ビットモードにおいては32ビットの仮想アドレスを，64ビットモードにおいては64ビットの仮想アドレス（TLBには40ビット分の領域しかないが）を，通常は36ビットの物理アドレスに変換する。この仕組みを図

図1　CP0レジスタとTLB

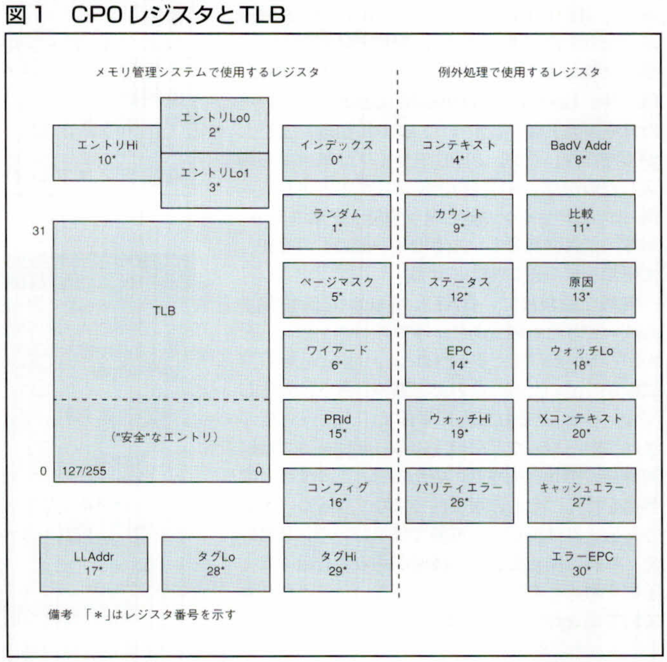


図2　TLBエントリの形式

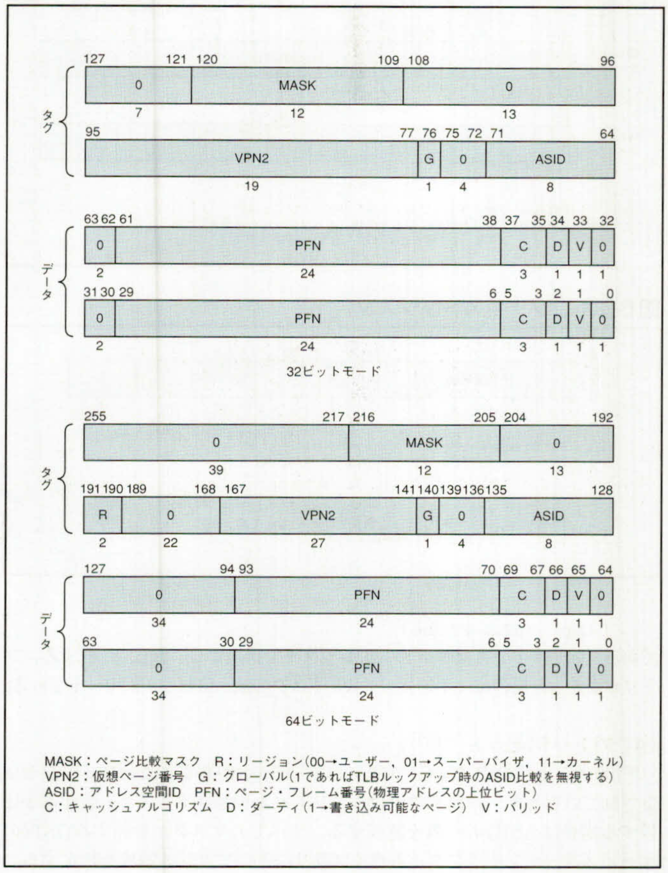
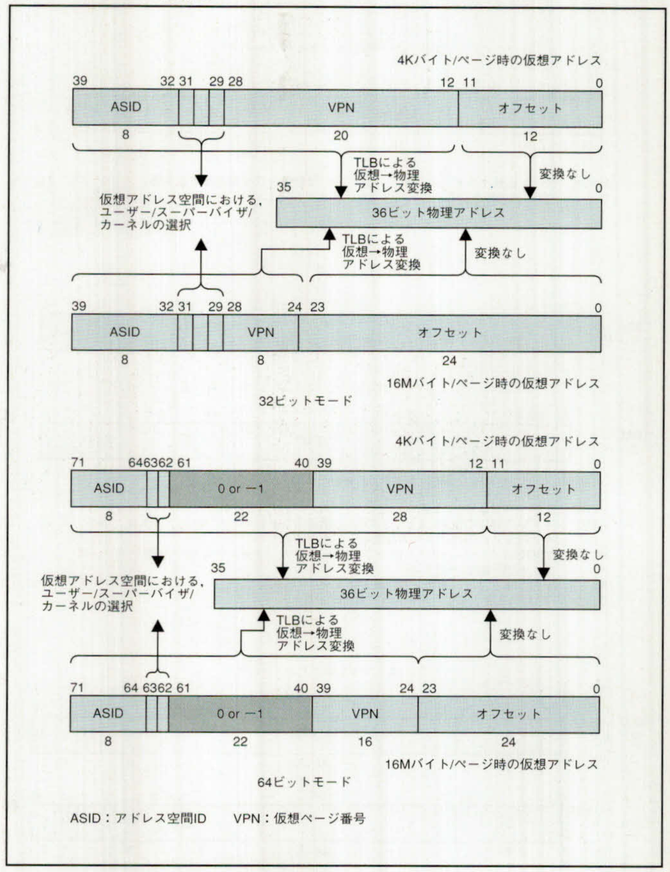


図3　アドレス変換



3に示す。

物理アドレスのビット数はMPUによって異なる。由緒正しいR4000系のMPUでは36ビットであり，R5000系のMPUもこれを継承している。その中間に開発されたR4200は33ビットという中途半端(？)なビット数である。後継のR4300はシステムバス幅が，従来の64ビットから32ビットに変更されたため，物理アドレスは32ビットしかサポートできない。ところが，R10000では物理アドレスが40ビットに拡張された。こうなると，エントリLo0/エントリLo1レジスタのビット長は32ビットでは不足するので，32ビットモード，64ビットモードにかかわらず，64ビット長のレジスタとなっている。

MMUのサポートするページサイズはエントリごとに4Kバイトから16Mバイトの範囲で4の倍数で指定できる。これはエントリへの書き込み時にページマスクレジスタで指定する。アドレス変換時に，仮想アドレス番号の下位ビットをページマスクレジスタの値で無視して仮想アドレスの検索が行われる。

TLBは48エントリ(R4200，R4300では32エントリ)のフルアソシアティブ構成で，1エントリは連続する2ページ分(偶数ページと奇数ページ)を示すひとつの仮想アドレスと，それに対応する2つの物理アドレスを保持している。仮想アドレスはエントリHiレジスタ，偶数ページ，奇数ページに対応する。物理アドレスは，それぞれ，エントリLo0レジスタ，エントリLo1レジスタで指定する。このTLB形態は一般にダブルエントリ形式と呼ばれている。48エントリではあるが，実質的には96エントリ相当，あるいは指定したページサイズの2倍のページサイズを持っているとみなせるためTLBのヒット率が高くなるといわれている。なお，エントリHiレジスタは仮想アドレスのほかにタスク番号に対応するASID(Address Space ID)を指定できる。エントリLo0，エントリLo1レジスタは物理アドレスのほかにキャッシュ情報，保護情報を指定できる。

(2)TLBの設定

TLBの設定に用いるレジスタは，エントリHi，エントリLo0，エントリLo1，ページマスク，インデックス(またはランダム)の5種類である。それぞれのレジスタ(インデックスとランダムレジスタを除く)の形式を図4に示す。TLBの各エントリの設定を行うためには，上述のレジスタに値を入れ，

```
tlbwi (TLB Write by Index Register)
```

または，

```
tlbwr (TLB Write by Random Register)
```

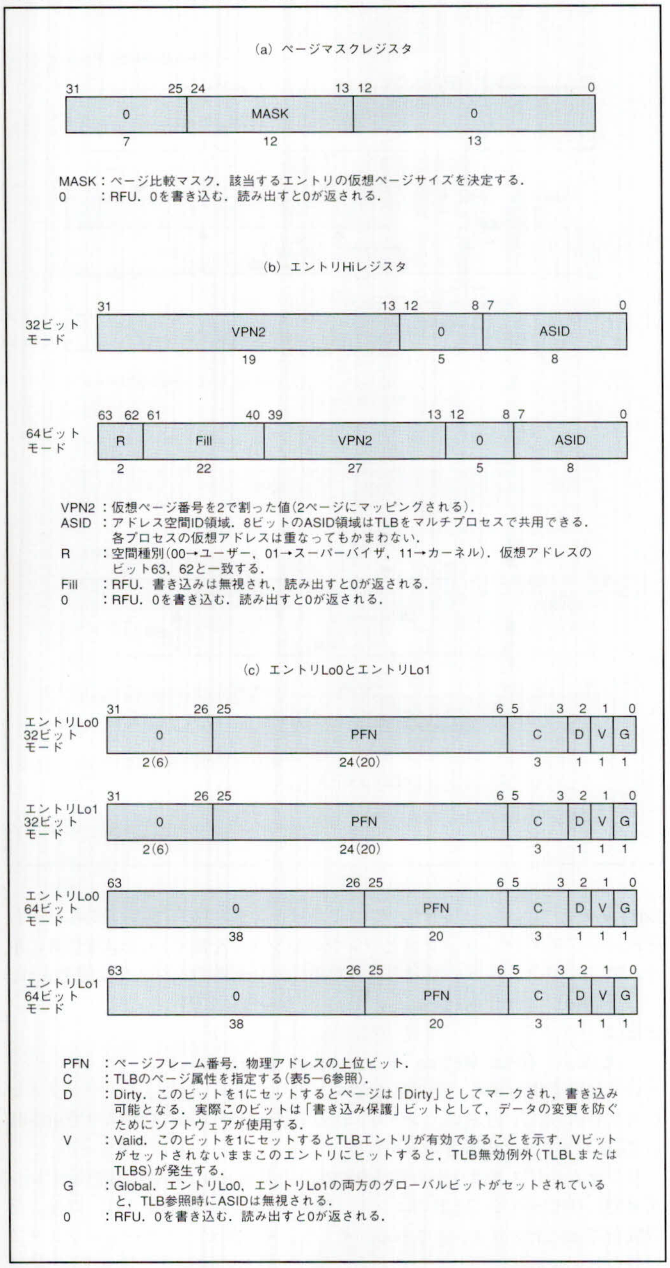
という命令を実行する。その名が示すとおり，tlbwi命令はインデックスレジスタで示されるTLBのエントリ，tlbwr命令はランダムレジスタで示されるTLBエントリへの書き込みを行う。

ランダムレジスタは0から(TLBのエントリ数－1)の間の値を示すレジスタで，リセットで(TLBのエントリ数－1)の値に初期化され，以後，命令実行ごとにデクリメントされる。そして，その値がワイアードレジスタに保持されている値に等しくなったら，再び(TLBのエントリ数－1)の値になる。tlbwr命令でTLBエントリへの書き込みを行う場合，番号が0から(ワイアードレジスタの値－1)の間のエントリ(これをワイアードエントリという)には書き込みができない。tlbwi命令ではワイアードエントリと無関係に書き込みができる。ワイアードエントリには，TLBの中から追い出されては困る，OS内のアドレス変換情報などを保持しておく。

TLBエントリの設定の具体例を示そう。ページサイズを4Kバイト，仮想アドレス0x2000(および，0x3000。ダブルエントリのため)に対応する物理アドレスを，偶数ページが0x333000，奇数ページが0x555000としよう。この情報をTLBのエントリ1に入れるものとする。このとき，ソフトウェアとしては(大体)次のようになる。ここで，asid，c，d，v，g，は適当な値を取るものとする。

```
mtc0    r0,C0_PageMask
li      r1,(0x2000|asid)
mtc0    r1,C0_EntryHi
li      r2,((0x333000>>6)|(c<<3)|(d<<2)|(v<<1)|g)
```

図4　TLBエントリ/レジスタ




```
    mtc0    r2,C0_EntryLo0
    li      r3,((0x555000>>6)|(c<<3)|(d<<2)|(v<<1)|g)
    mtc0    r3,C0_EntryLo1
    li      r4,1
    mtc0    r4,C0_Index
    tlbwi
```

(3) TLBの参照

TLBの各エントリの内容を参照することもできる。インデックスレジスタで指定されるエントリの内容を，ページマスク，エントリHi，エントリLo0，エントリLo1レジスタに読み出す命令が，

```
    tlbr  (TLB Read)
```

である。逆に，指定する仮想アドレスに対応する，変換情報が格納されているTLBエントリの番号をインデックスレジスタに書き込む命令が，

図5　コンテキストレジスタ

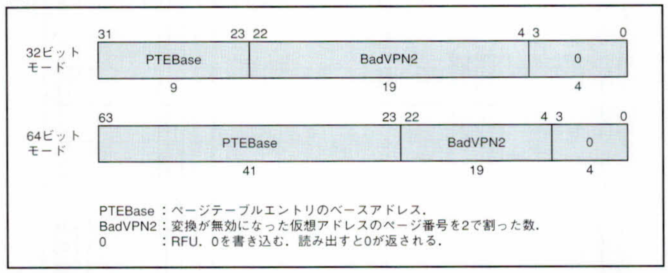
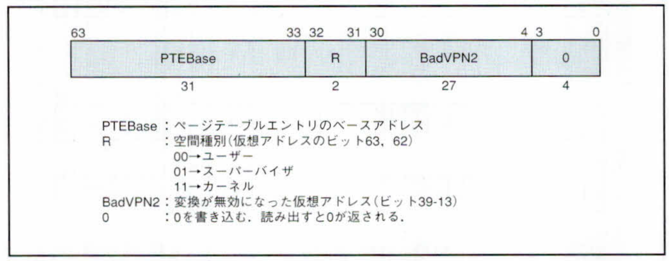


図6　Xコンテキストレジスタ




```
    tlbp  (TLB Probe)
```

である。仮想アドレスはエントリHiレジスタで指定する。もし，一致するエントリがなかった場合は，インデックスレジスタの最上位ビットがセットされる。

(4) グローバルビット

TLBエントリの中にはG（グローバル）ビットがある。このビットがセットされているとき，TLBのタグ部が与えられた仮想アドレスと一致するか比較する場合にASIDの一致を無視する。すべてのタスクが参照するOS内のサービスルーチンやランタイムライブラリなどのアドレス領域を指定するのに使用する。GビットはTLBエントリの中には1ビットしかないが，エントリLo0レジスタとエントリLo1レジスタの両方にGビットを設定する領域がある。両方のGビットの指定が1のときのみ，TLBエントリ内のGビットがセットされる。

(5) コンテキストレジスタ

アドレス変換において，仮想アドレスと物理アドレスの対応表はメインメモリ内に置かれている。これをページテーブルと呼び，タスクごとに別々の組み合わせを持っている。MIPS系のMPUにおいては，TLBミスが発生した場合は，メインメモリをアクセスしてミスした仮想アドレスに対応する物理アドレスをページテーブルの中から読み出してきて，その情報をTLBに設定しなければならない。その場合に利用するレジスタがコンテキストレジスタである。図5にコンテキストレジスタの形式を示す。

コンテキストレジスタは2つの領域からなる。読み書き可能なPTEBase領域と，読み出し専用のBadVPN2領域である。PTEBaseにはタスク切り替え時に，そのタスクのページテーブルのベースアドレスを設定する。TLBミスが発生したとき，その仮想アドレス（のページ番号を2で割った値。仮想アドレスのビット31-13）がBadVPN2領域に，自動的に設定される。このため，コンテキストレジスタは，ページサイズが4Kバイトの場合は，8バイトのページテーブルエントリのペア（ひとつの仮想アドレスに対して2つの物理アドレスが必要なため，合計で16バイト）を参照するポインタとしてそのまま使用できる。ページサイズが4Kバイトでない場合は，このレジスタの値を適当にシフトしたりマスクしたりして，ページテーブルエントリへのポインタを生成することができる。コンテキストレジスタは32ビットモードと64ビットモードの2つが定義されているが，アドレス空間が64ビットモードの場合はXコンテキストレジスタ（図6）のほうを使用する。

(6) TLB不一致処理の概略

TLB例外（TLBミス，TLB無効，TLB変更）が発生した場合，例外を発

生した仮想アドレスは，エントリHi，BadVAddr，コンテキストレジスタに自動的に設定される。このとき，OSでの処理はだいたい次のような感じになる。

```
    mfc0     r1,C0_Context
    lw       r2,0(r1)          // 偶数ページの物理アドレス
    lw       r3,8(r1)          // 奇数ページの物理アドレス
    tlbp                       // TLB内にある仮想アドレス？
    mfc0     r1,C0_Index
    bltz     r1,NOTFOUND       // なければ分岐
    nop
    tlbr                       // あれば内容を読み出す

        適当な処理

    mtc0     r2,C0_EntryLo0    // 物理アドレス情報を更新
    mtc0     r3,C0_EntryLo1    // 物理アドレス情報を更新
    tlbwi                      // 同じエントリにライト
    j        COMMON
    nop
NOTFOUND:
    mtc0     r2,C0_EntryLo0    // 物理アドレス情報を新設
    mtc0     r3,C0_EntryLo1    // 物理アドレス情報を新設
    tlbwr                      // ランダムにライト
COMMON:
```

キャッシュ

(1)キャッシュの構成

MIPS系のMPUにおいてキャッシュの存在は必須である。ただし，その構成については特に規定がない。たとえば，R3000/R4000はダイレクトマップ，R5000/R10000は2ウェイセットアソシアティブ，Rubyは(おそらく)4ウェイセットアソシアティブである。最近発表されたJadeやOpalは1～4セットアソシアティブを設計時にユーザーが選択できるようになっている。1次キャッシュのアドレスの比較方式は，R3000が物理アドレスキャッシュ，それ以外が，仮想アドレスインデックス，物理タグ比較である。開発中止になったBeastというMPUは仮想アドレスキャッシュだったという噂である(真相は誰にもわからない)。

2次キャッシュに関しては，MPUに外付けのため，物理アドレスキャッシュである。R4000では2次キャッシュを命令とデータ用に2分割できる仕組みを持っていた。しかし，この機能はバグが取り切れなかったため廃止になった(と記憶している)。以降，2次キャッシュは命令とデータが混合したユニファイドキャッシュ構成1本に絞られた。キャッシュの書き込み制御はR3000がライトスルー，それ以外は，基本的にライトバック方式である。R5000やR5400など，TLBのC領域でライトスルーを指定できるものもあるが，あまり利用されていない。ダイレクトマップ構成の1次キャッシュと2次キャッシュの動作を，それぞれ図7，図8に示しておく。

さて，MIPS系のMPUにおいて，1次キャッシュの容量と構成については図9のようになっている。R3000に関しては1次キャッシュが外付けなので大容量のキャッシュをサポートできる。最近の組み込み系のMPUでは命令キャッシュ16Kバイト，データキャッシュ8Kバイトというのが流行である。なぜか，データキャッシュの容量が命令キャッシュの容量の半分であることが多い。なお，実装されるキャッシュ容量に関してはコンフィグレジスタ(図10)のIC領域，DC領域でソフトウェアから参照できる。このほかにもコンフィグレジスタはMPUを性格付けるさまざまな情報を参照したり設定するために用いられる。コンフィグレジスタの形式はMPUごとに異なるが，R4000系，R5000系のMPUでは似たような形式をしているのがわかる。R10000系に関しては内容がかなり異なるので図10では示していない。

(2)キャッシュ命令

MIPS系のMPUではキャッシュを操作するためにキャッシュ命令が命令セットとして定義されている。キャッシュ命令はキャッシュのラインを無効化したり，ダーティなラインをメインメモリへ強制的にライトバックしたりできる。キャッシュ命令の形式は，

```
    cache      機能コード,offset(base)
```

となっている。baseで示されるベースレジスタの内容にoffsetの値を加えて生成される仮想アドレスによって，キャッシュを参照する。機能コードの種

図7　1次キャッシュの動作(ダイレクトマップ方式)

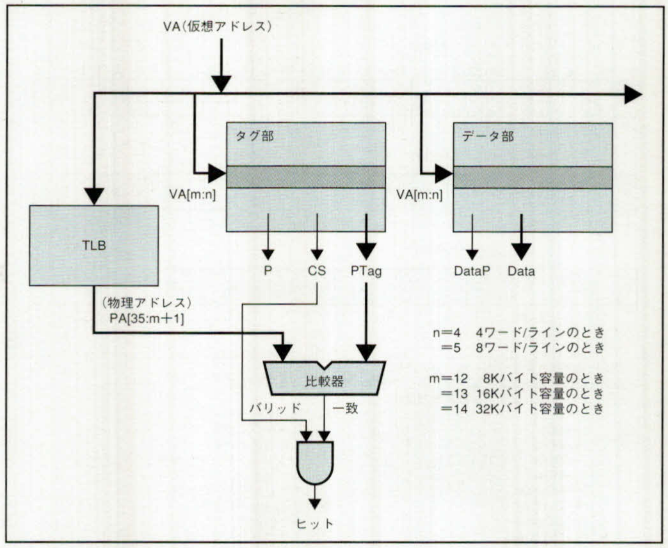


図8　2次キャッシュの動作

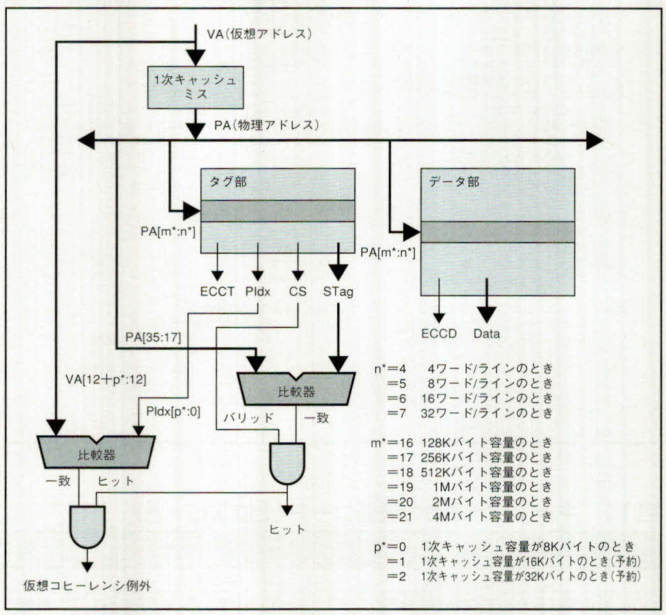


図9　MIPS系MPUのキャッシュ容量と構成

MPU	命令キャッシュ	データキャッシュ	キャッシュの構成
R3000	4K-256KB	4K-256KB	ダイレクトマップ
R4000	8KB	8KB	ダイレクトマップ
R4400	16KB	16KB	ダイレクトマップ
R4300	16KB	8KB	ダイレクトマップ
R5000	32KB	32KB	2ウェイセット
R5400	32KB	32KB	2ウェイセット
R10000	32KB	32KB	2ウェイセット
R16000	64KB?	64KB?	2ウェイセット?

図10 コンフィグレジスタ

R4000

CM	EC	EP	SB	SS	SW	EW	SC	SM	BE	EM	EB	0	IC	DC	IB	DB	CU	K0
31	30–28	27–24	23–22	21	20	19–18	17	16	15	14	13	12	11–9	8–6	5	4	3	2–0
1	3	4	2	1	1	2	1	1	1	1	1	1	3	3	1	1	1	3

R5000

0	EC	EP	SB	SS	EW	SC	1	BE	EM	EB	SE	IC	DC	IB	DB	0	K0
1	3	4	2	2	2	1	1	1	1	1	1	3	3	1	1	1	3

VR4111

0	EC	EP	AD	0	M16	0	1	0	BE	1	0	CS	IC	DC	0	0	0	K0
1	3	4	1	2	1	2	1	1	1	1	1	1	3	3	1	1	1	3

CM…マスターチェッカーモードの許可
EC…システムインタフェイスクロックの周波数比(プロセッサ依存)
EP…ライトバックデータの転送パターン
　0→D
　1→DDx
　2→DDxx
　3→DxDx
　4→DDxxx
　5→DDxxxx
　6→DxxDxx
　7→DDxxxxx
　8→DxxxDxxx
　その他→RFU
SB…2次キャッシュラインサイズ
　0→ 4ワード
　1→ 8ワード
　2→16ワード
　3→32ワード
SS(R4000)…分割2次キャッシュモード
　0→ユニファイドキャッシュ
　1→命令/データ分割キャッシュ
SS(R5000)…2次キャッシュサイズ
　0→512Kバイト
　1→1Mバイト
　2→2Mバイト
　3→2次キャッシュなし
SW…2次キャッシュのデータ幅
　0→128ビット
　1→ 64ビット
EW…システムバス幅
　0→64ビット
　1→32ビット
SC…2次キャッシュの有無
　0→あり
　1→なし
SM…ダーティシェアド状態の禁止
　0→ダーティシェアド状態許可
　1→ダーティシェアド状態禁止
AD…高速データモード
　0→R4000互換
　1→高速モード
BE…ビッグエンディアン状態
　0→リトルエンディアン
　1→ビッグエンディアン
EM…ECCモードの許可
　0→ECCモード
　1→パリティモード
EB…リフィル時のオーダリング
　0→シーケンシャル
　1→サブブロック
SE…2次キャッシュの許可
　0→禁止
　1→許可
CS…キャッシュサイズモード
　0→$2^{(12+n)}$バイト
　1→$2^{(10+n)}$バイト
IC…命令キャッシュサイズ(1次)
　0→ 4Kバイト(CS=0)，　1Kバイト(CS=1)
　1→ 8Kバイト(CS=0)，　4Kバイト(CS=1)
　2→16Kバイト(CS=0)，　8Kバイト(CS=1)
　3→32Kバイト(CS=0)，16Kバイト(CS=1)
　4→64Kバイト(CS=0)，32Kバイト(CS=1)
DC…データキャッシュサイズ(1次)
　意味はICと同じ
IB…1次命令キャッシュラインサイズ
　0→4ワード
　1→8ワード
DB…1次データキャッシュラインサイズ
　0→4ワード
　1→8ワード
CU…SC命令での更新
　0→SC命令はTLBで指定したキャッシュアルゴリズムを使用
　1→SC命令はライト時コヒーレントアップデートアルゴリズムを使用
K0…Kseg0のキャッシュアルゴリズム
　TLBのC領域と同じ

図11 キャッシュ命令の機能コード(上位3ビット)

コード	キャッシュ	機能の名称	操作
0	I,SI	Index Invalidate	キャッシュラインの無効化
0	D,SD	Index Writeback Invalidate	キャッシュラインがダーティならライトバックし無効化
1	All	Index Load Tag	キャッシュラインのタグの値をタグLoレジスタに格納
2	All	Index Store Tag	タグLoレジスタの値をキャッシュラインのタグに設定
3	D,SD	Create Dirty Exclusive	キャッシュミスのとき，前のラインがダーティならライトバックする。タグを設定し直し状態をダーティにする
4	All	Hit Invalidate	ヒットするラインの無効化
5	D,SD	Hit Writeback Invalidate	ヒットするラインがダーティならライトバックし無効化
5	I	Fill	キャッシュラインに命令を読み込む
6	I,D,SD	Hit Writeback	命令キャッシュではラインをライトバックする。データキャッシュではラインがダーティならライトバックする
7	SI,SD	Hit Set Virtual	2次キャッシュの仮想インデックスの変更

類はMPUによって異なる。図11にR4000系のMPUのキャッシュ命令で使用される機能コードを示す。機能コードは5ビットからなり，下位2ビットが操作対象になるキャッシュの種類，上位3ビットが操作の種類を示す。キャッシュの種類は次のようになっている。

0 = (I) 1次命令キャッシュ
1 = (D) 1次データキャッシュ
2 = (SI) 2次命令キャッシュ
3 = (SD) 2次データキャッシュ

なお，2次キャッシュはユニファイドキャッシュとして構成されるので，SIを指定することはほとんどなく，たいていはSDを指定してキャッシュの操作を行う。

(3)キャッシュの初期化

MIPS系のMPUの特徴は「キャッシュは自動的には初期化されない」ということである。つまり，リセット後にキャッシュをキャッシュ命令を使って初期化しなければ，キャッシュを使用することができない。リセットで自動的にキャッシュが初期化されるという，ほかのMPUでの経験がある人には奇異に思えるかもしれないが，そこはRISCということで納得してもらおう。

ここでは1次キャッシュを初期化する手順について述べる。2次キャッシュを有するMPUでは当然2次キャッシュの初期化も行わなければならないが，手順は1次キャッシュと同様なので省略する。

キャッシュの初期化とは全ラインのタグ部を無効化することである。そのためにはキャッシュ命令を用いて各ラインのタグ部に0を書き込めばよい。前準備として，まずキャッシュの容量とラインサイズを知らなければならない。これは，コンフィグレジスタのCS(ビット12)，IC(ビット11-9)，DC(ビット8-6)，IB(ビット5)，DB(ビット4)領域を参照すればよい。

CSはキャッシュ容量が4Kバイトよりも小さなMPUのためにNECのVR4100シリーズで新設されたビットである。VR4100シリーズではCSの値は1固定となっている。ほかのR4000系のMPUでは0である(R5000系はCSの位置に別のビットが割り当てられているが)。図10の図中の説明にもあるが，CSの値が0ならキャッシュ容量は，命令キャッシュが2の(12＋IC)乗バイト，データキャッシュが2の(12＋DC)乗バイトである。CSの値が1ならキャッシュ容量は，命令キャッシュが2の(10＋IC)乗バイト，データキャッシュが2の(10＋DC)乗バイトである。IB，DBに関しては0のときが4ワード(16バイト)，1のときが8ワード(32バイト)である。これらを考慮すると(1次)キャッシュの初期化(R4000系)は次のような命令列で行える。

```
    mtc0   r0,C0_TagLo
    mfc0   r1,C0_Config
    srl    r2,r1,9
    andi   r2,r2,0x7       // IC領域
    srl    r3,r1,6
    andi   r3,r3,0x7       // DC領域
    li     r4,16
    andi   r9,r1,0x20      // IB領域
    bnel   r9,r0,1f        // Brach Likely
    addu   r4,r4,r4        // IBが0ならスキップ
1:
    li     r5,16
    andi   r9,r1,0x10      // DB領域
    bnel   r9,r0,2f        // Brach Likely
    addu   r5,r5,r5        // DBが0ならスキップ
2:
    li     r4,12
    andi   r1,r1,0x1000    // CSビット
    bnel   r1,r0,3f        // Brach Likely
    li     r4,10           // CSが0ならスキップ
3:
    li     r1,1
    sllv   r6,r1,r2        // 命令キャッシュの容量
    sllv   r7,r1,r3        // データキャッシュの容量
    li     r1,0x80000000 // キャッシュ領域の開始アドレス
    addu   r2,r1,r6        // キャッシュ領域の終了アドレス
4:
    cache  Index_Store_Tag_I,0(r1)
    addu   r1,r1,r4        // ラインサイズを加算
    bne    r1,r2,4b
```

```
    nop
    li      r1,0x80000000 // キャッシュ領域の開始アドレス
    addu    r2,r1,r7      // キャッシュ領域の終了アドレス
5:
    cache   Index_Store_Tag_D,0(r1)
    addiu   r1,r1,r5      // ラインサイズを加算
    bne     r1,r2,5b
    nop
```

(4) インデックスによるウェイの指定方法

キャッシュ命令で，キャッシュのインデックスを指定して処理を行う場合，ダイレクトマップ方式のキャッシュでは，指定した仮想アドレスの一部がそのままインデックスとして用いられる。しかし，2ウェイセットアソシアティブ方式のキャッシュにおいては，インデックスに加え，ウェイを指定しなければ，キャッシュラインが一意に定まらない。このウェイの指定には2とおりの方法がある。

(i) R5000方式

キャッシュ容量が2のn乗バイトであるとき，仮想アドレスのビット(n－1)の値でウェイを指定する。総エントリのうち，前半がウェイ0，後半がウェイ1に対応する。R5000ではキャッシュ容量が32Kバイト(n＝15)であるから，仮想アドレスのビット14がウェイの指定になる。この方式だと，キャッシュ構成が2ウェイになったからといって，ダイレクトマップ用から，キャッシュの初期化のプログラムを変更する必要はない。欠点としては，たとえば，

```
    cache   Index_Store_Tag_D,0x4000(r10)
```

などによって，ウェイ1を指定したつもりでも，r10が示すベースアドレスのビット14の値が1であれば，オフセットとベースアドレスの加算によって生成される実効アドレスのビット14は0になってしまうので，ウェイ0のほうを指定したことになる。また，ベースアドレスが，アドレス範囲の境界を示しているような場合は，0x4000というオフセットを加算することで桁の繰り上がりが発生し，アドレスエラーを引き起こすような実効アドレスが生成されてしまうこともある。これらの状況を避けるためには，ベースアドレスのビット14の値をあらかじめ0にしておく，などのプログラム処理が必要になる。

(ii) R10000方式

指定する仮想アドレスの最下位ビットでウェイを指定する。たとえば，

```
    cache   Index_Store_Tag_D,0x1(r10)
```

によって，確実にウェイ1を指定できる。なぜなら，キャッシュ命令に与えられる仮想アドレスは4の倍数であることが保証されているので，少なくとも下位2ビットは0であり，1を加算しても桁の繰り上がりは発生しないためである。欠点としては，キャッシュの初期化でウェイ0と1を別個に指定するプログラムを書かなければならない点くらいであろうか。

アドレス空間

(1) 保護モード

R4000以降のMPUではプログラム実行の保護レベルを，カーネル，スーパーバイザ，ユーザーの3種類に分類している。カーネルモードが最高の特権性を持ち，スーパーバイザモードがその次，ユーザーモードが最低の特権性となっている。

OSの実行はカーネルモードで実行され，アプリケーションプログラムはユーザーモードで実行される。スーパーバイザモードはその中間で，デバッガやデバイスドライバなどの実行を期待していると思われるが，実際のOSでこのモードが使用されているかは疑問である。マニュアルには，スーパーバイザモードは，OSの実行を階層化する目的，つまり，カーネル部分は，まさにカーネルモードで実行し，OSのそれ以外の部分はスーパーバイザモードで実行する目的で導入されたと記述してある。個人的には，カーネルモードとユーザーモードの2種類で十分ではないかと思っている。実際，R3000ではこの2種類のモードしかなかった。また，例外や割り込みの発生時はカーネルモード(に相当するモード)に遷移するため，そこからスーパーバイザモードに移行するにはかなり意図的なプログラム処理が必要である。

(2) 32ビットモード/64ビットモード

R4000以降では，64ビットプロセッサということで，仮想アドレス空間が64ビットに拡張された。64ビットの仮想空間のうち，アプリケーション(ユーザーモード)で使用できる空間は，物理アドレスが36ビット(4Gバイト，MPUによって若干異なる)であり，仮想アドレスは，32ビットモードで31ビット(2Gバイト)，64ビットモードで40ビット(1Tバイト)である。

図12，図13，図14に，それぞれ，ユーザーモード，スーパーバイザモード，カーネルモードでのアドレス空間を示す。図中で「TLBマッピング」というのはアドレス変換が可能な領域を表している。この領域がキャッシュ可能か不可能かはTLBエントリのC(キャッシュアルゴリズム)領域で指定する。この領域の値が2のとき，キャッシュ不可である。それ以外はキャッシュ可能であり，マルチプロセッサ構成時にいろいろなキャッシュ動作を指定できるようになっているが，詳細は省く。

アドレス変換を行わない領域は，アドレスの値によって，キャッシュの可，不可が分かれている。リセット直後では，まだTLBやキャッシュの初期化が行われていないのでMMUやキャッシュを使用できないため，TLBマップなしのキャッシュ不可領域から実行が始まるようになっている。TLBマップを行わない領域の仮想アドレスと物理アドレスの対応は，kseg0，kseg1，ckseg0，ckseg1領域では仮想アドレスのビット28-0がそのまま物理アドレ

図12 ユーザーモードアドレス空間

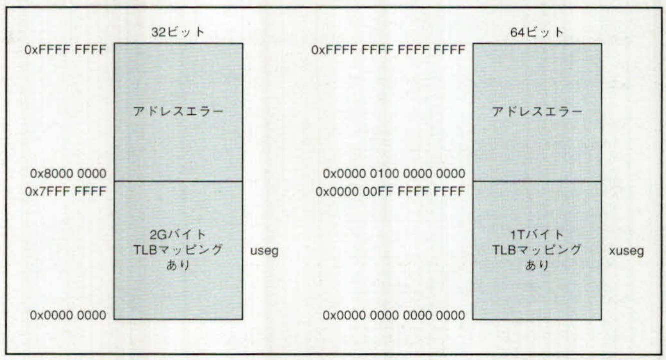


図13 スーパーバイザモードアドレス空間

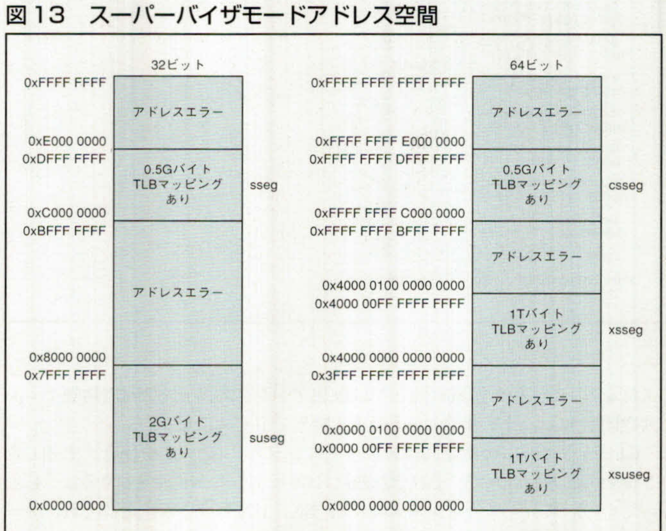
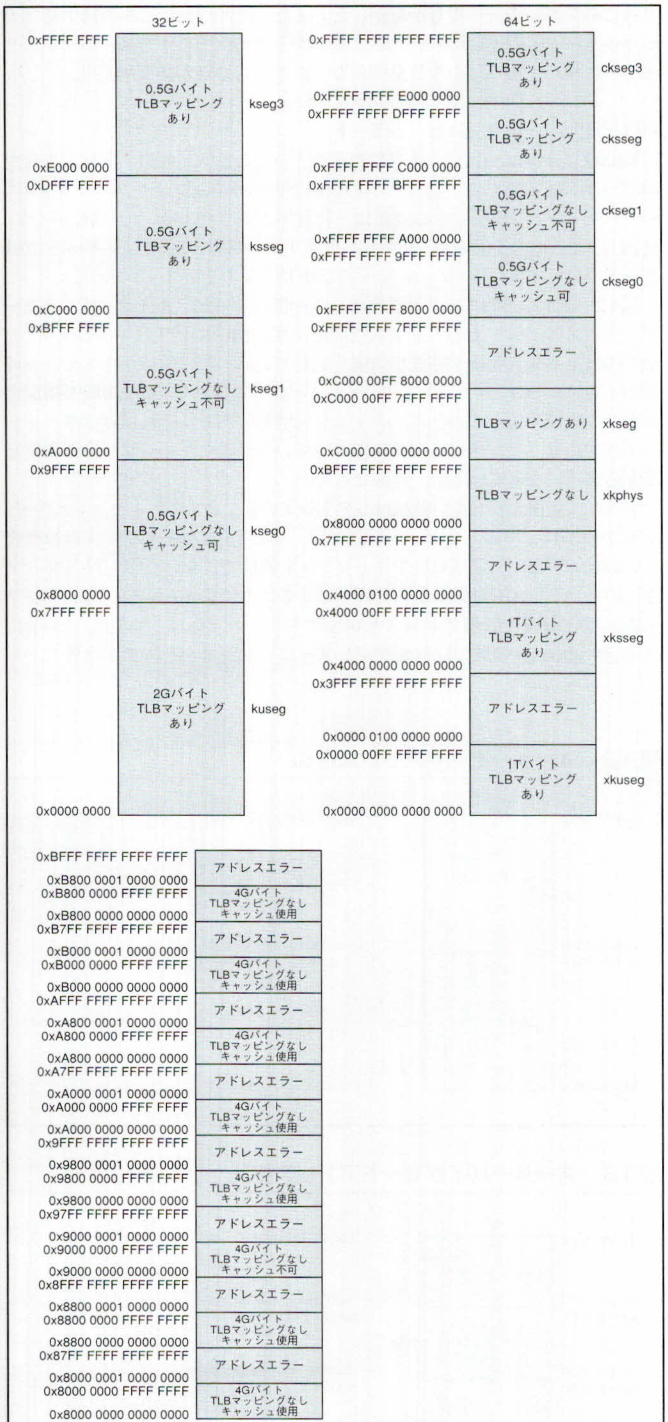


図14　カーネルモードアドレス空間



注)R4300の場合の図．一般にはVA[61:59]がキャッシュアルゴリズムを示している．

図15　ステータスレジスタ

31–28	27	26	25	24–16	15–8	7	6	5	4–3	2	1	0
CU(CU3:CU0)	RP	FR	RE	DS	IM(7:0)	KX	SX	UX	KSU	ERL	EXL	IE
4	1	1	1	9	8	1	1	1	2	1	1	1

CU　：4つのコプロセッサの使用許可(1→使用可，0→使用不可)．
CP0はカーネルモードにおいてCU0ビットにかかわらず常に使用可能．
CP2は将来の拡張用に予約されている．
CP3はMIPS IV命令の許可ビット．
RP　：低電力モードの許可(0→通常，1→低電力モード)．
内部クロック周波数，システムインタフェイスクロックの周波数を通常の1/4にする．
FR　：使用可能な浮動小数点レジスタ数(0→16本，1→32本)．
RE　：ユーザーモードにおけるエンディアン反転の許可(0→禁止，1→反転)．
DS　：自己診断ステータス領域．
IM　：割り込みマスク．外部，内部，コプロセッサおよびソフトウェア割り込みの許可(0→禁止，1→許可)．
8ビットで構成され，8つの割り込みを制御する．
割り込みは各ビットに次のように割り当てられている．
IM7　：タイマ割り込みのマスク
IM(6:2)　：外部通常割り込み，Int(4:0)および外部ライトリクエストのマスク
IM(1:0)　：ソフトウェア割り込み，原因レジスタIP(1:0)のマスク
KX　：カーネルモードでの64ビットアドレシングの許可(0→32ビット，1→64ビット)．
このビットがセットされていた場合，カーネルモードアドレス空間でのTLBミス時，XTLB不一致例外が発生する．
なお，カーネルモードでは，64ビットオペレーションは常に有効．
SX　：スーパーバイザモードでの64ビットアドレシングと64ビットオペレーションの許可(0→32ビット，1→64ビット)．
このビットがセットされていた場合，スーパーバイザモードアドレス空間でのTLBミス時，XTLB不一致例外が発生する．
UX　：ユーザーモードでの64ビットアドレシングと64ビットオペレーションの許可(0→32ビット，1→64ビット)．
このビットがセットされていた場合，ユーザーモードアドレス空間でのTLBミス時，XTLB不一致例外が発生する．
KSU：オペレーティングモードの設定／表示(10→ユーザー，01→スーパーバイザ，00→カーネル)．
ERL：エラーレベルの設定／表示(0→正常，1→エラー)．
EXL：例外レベルの設定／表示(0→正常，1→例外)．
IE　：割り込み許可の設定／表示(0→禁止，1→許可)．

DS領域(R4300の場合)

24	23	22	21	20	19	18	17	16
ITS	0	BEV	TS	SR	0	CH	CE	DE
1	1	1	1	1	1	1	1	1

ITS　：命令トレースサポート(0→禁止，1→許可)．
BEV　：TLB不一致例外ベクタと汎用例外ベクタのベースアドレスの指定(0→通常，1→ブートストラップ)．
TS　：TLBシャットダウンの発生(読み出し専用)(0→未発生，1→発生)．
SR　：ソフトリセットまたはNMIの発生(0→未発生，1→発生)．
CH　：CP0の条件ビット(0→偽，1→真)．ソフトウェアでだけ読み書き可能で，ハードウェアではアクセスできない．
CE，DE：これらのビットはR4200との互換性を保つために定義している．
R4300のハードウェアでは使用しない．
0　：RFU．0を書き込む．読み出すと0が返される．

スになる。xkphys領域に関しては仮想アドレスのビット35-0(物理アドレス空間が36ビットの場合)がそのまま物理アドレスになる。

64ビットモードの指定にはステータスレジスタ(図15)を使う。上述した3種類の保護レベルで，それぞれ独立に64ビットモード設定できるようになっている。ステータスレジスタのKX，SX，UXビットが，それぞれ，カーネル，スーパーバイザ，ユーザーモードに対応し，これらのビットをセットすることで64ビットモードになる。64ビットモードを指定すると，64ビット命令と64ビット仮想アドレス空間を使用できるようになる。ただし，64ビット命令はカーネルモードでは常時使用可能である。カーネルモードでの64ビットモードの指定とは64ビット仮想アドレス空間を使用するという意味しか持っていない。

例外割り込み処理

(1)例外割り込みの種類

MPUは(当然ながら)さまざまな例外や割り込みを処理できる。例外や割り込みが発生すると，その要因が原因レジスタ(図16)の中にセットされ，要因を発生した命令の仮想アドレスがEPCレジスタに格納される。そのあと，あらかじめ定められたアドレス(例外ベクタ)へ分岐する。図17に原因レジスタに設定される例外コードの一覧を示す。

(2)例外ベクタ

例外や割り込みが発生すると，例外の種類とステータスレジスタのBEVビットの値によって，ある決められたアドレス(例外ベクタ)に分岐する。例外ベクタのアドレス一覧を図18に示す。BEVビットはブートストラップ，すなわち，リセット直後でTLBやキャッシュの初期化ができていない状態を表している。このため，BEVビットがセットされているときの例外ベクタは，TLBマップなし，キャッシュ不可領域となっている。BEVビットがク

図16　原因レジスタ

31	30	29–28	27–16	15–8	7	6–2	1–0
BD	0	CE	0	IP(7:0)	0	ExcCode	0
1	1	2	12	8	1	5	2

BD　：最後に発生した例外が分岐遅延スロット内で実行されたかどうかを示す(1→遅延スロット内，0→通常)．
CE　：コプロセッサ使用不可例外が発生したコプロセッサ番号を示す．この例外が発生していないときは不定．
IP　：ペンディングされている割り込みを示す(1→保留中，0→割り込みなし)．
割り込みは各ビットに次のように割り当てられている．
- IP7　：タイマ割り込み
- IP(6:2)　：外部通常割り込み，$\overline{\text{Int}}$(4:0)および外部ライトリクエストで制御．
- IP(1:0)　：ソフトウェア割り込み．このビットだけソフトウェアで1にセットすると割り込み例外を発生する．

ExcCode：例外コード領域．
0　：RFU．0を書き込む．読み出すと0が返される．

図17　原因レジスタに設定される例外コード

例外コード	略号	説明
0	Int	割り込み
1	Mod	TLB変更例外
2	TLBL	TLB不一致例外(ロード，命令フェッチ)
3	TLBS	TLB不一致例外(ストア)
4	AdEL	アドレスエラー(ロード，命令フェッチ)
5	AdES	アドレスエラー(ストア)
6	IBE	バスエラー(命令フェッチ)
7	DBE	バスエラー(ロード，ストア)
8	Sys	システムコール
9	Bp	ブレークポイント
10	RI	予約済み命令例外
11	CpU	コプロセッサ使用不可例外
12	Ov	演算オーバーフロー例外
13	Tr	トラップ
14	VCEI	命令仮想コヒーレンシ例外
15	FPE	浮動小数点演算例外
16−22		未使用
23	WATCH	ウオッチ例外
24−30		未使用
31	VCED	データ仮想コヒーレンシ例外

図18　例外ベクタのアドレス

例外・割り込みの種類	アドレス
リセット・NMI	0xFFFFFFFFBFC00000
キャッシュエラー	0xFFFFFFFFA0000100 (BEV=0)
	0xFFFFFFFFBFC00300 (BEV=1)
TLB不一致(ミス)	0xFFFFFFFF80000000 (BEV=0)
(EXL=0)	0xFFFFFFFFBFC00200 (BEV=1)
X TLB不一致(ミス)	0xFFFFFFFF80000080 (BEV=0)
(EXL=0)	0xFFFFFFFFBFC00280 (BEV=1)
その他	0xFFFFFFFF80000180 (BEV=0)
	0xFFFFFFFFBFC00380 (BEV=1)

リアされているときは，TLBマップなし，キャッシュ可能領域である。BEVビットは，リセット直後にはセットされており，種々の初期化が完了したらソフトウェアでクリアする。

キャッシュエラーに関しては，それが発生するときには，もはやキャッシュは意味をなさないので，BEVビットとは無関係にTLBマップなし，キャッシュ不可領域に分岐する。BEVが0のときの例外発生時には仮想アドレス空間のkseg0領域が例外ベクタになるが，この領域はコンフィグレジスタのK0領域の値を2に設定することでキャッシュ不可に指定できる。なお，図18では64ビットアドレスで記述してあるが，32ビットモード時もMPUの内部では32ビットアドレスを符号拡張して64ビットアドレスとして処理しているので，32ビットモードでも同じアドレスになる。

さて，MIPS系のMPUでよく非難の対象になるのがこの例外ベクタである。致命的な例外とTLBミス以外(一般例外という)は，すべて同じベクタが使用される。そのとき，例外も割り込みも区別がない。例外ハンドラの先頭では，まず原因レジスタを読み，発生している例外や割り込みの要因を切り分けてからそれぞれの処理に移らなければならない。

また，割り込みに関してはレベルがひとつしかない。サポートされる割り込みは，外部割り込み(5種類)，タイマ割り込み，ソフトウェア割り込みがあるが，これらのどれが発生しても一様に一般例外用のベクタが使用される。プログラムによる要因の切り分けにいくつかの処理が必要になるうえ，EXLビットがセットされているときは割り込みがマスクされるので，割り込み応答性がよいとは決していえない。リアルタイム処理が必要な応用分野においては不利になる。

(3)EXLとERL

ステータスレジスタの中にEXL，ERLというビットがある。それぞれは，例外レベル，エラーレベルを表し，現在，例外(割り込み)やエラーが発生している状態にあることを意味する。

リセット，NMI，キャッシュエラーの発生時にはERLビットがセットされ，そのとき実行していたアドレスがエラーEPCレジスタに格納される。それ以外の例外ではEXLビットがセットされ，そのとき実行していたアドレスはEPCレジスタに格納される。EXLまたはERLビットがセットされるとMPUはカーネルモードと同じ特権性を持つようになる。たとえば，ユーザーモードでプログラムを実行している場合に例外や割り込みが発生すると，保護レベルはユーザーモードのままであるが，EXLビットがセットされることにより，OSでのプログラム処理が実行できるようになる。割り込みや例外処理から元のプログラムへの復帰には，

```
eret   (Return form Exception)
```

という命令を使用する。eret命令は，ERLビットがセットされていればエラーEPCレジスタから，EXLビットがセットされていればEPCレジスタからアドレスを取り出し，そこへ分岐する。そのときERL，またはEXLビットはクリアされる。

ところで，ERLビットには特別な機能があり，このビットがセットされていると，ユーザーアドレス領域は2GバイトのTLBマップなし(仮想アドレス＝物理アドレス)，キャッシュ不可領域になる。これは例外ハンドラ内でキャッシュエラー(ECCエラーやパリティエラー)を処理するための機能である。

(4)例外処理シーケンス

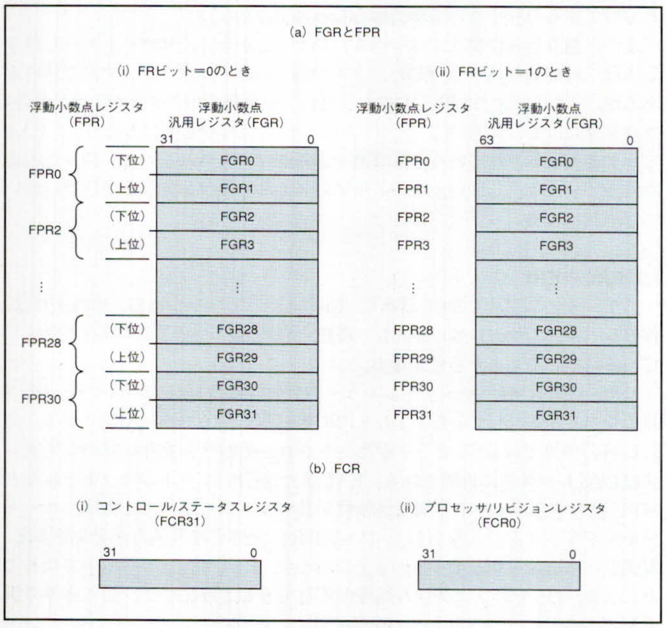
以下に，例外処理プログラムのシーケンスを簡単に示す。

```
mfc0   r1,C0_Cause
andi   r1,r1,0xff         // 例外コードを抽出
li     r2,(0<<2)
beq    r1,r2,Proc_Int     // 割り込み処理
li     r2,(1<<2)
beq    r1,r2,Proc_Mod     // TLB変更例外処理
li     r2,(2<<2)
beq    r1,r2,Proc_TLBL    // TLB不一致例外処理
li     r2,(3<<2)
beq    r1,r2,Proc_TLBS    // TLB不一致例外処理
li     r2,(4<<2)
beq    r1,r2,Proc_AdEL    // アドレスエラー処理
    :
    :

Proc_Int:
    適当な割り込み処理
j      COMMON
nop

Proc_Mod:
    適当なTLB変更例外処理
j      COMMON
nop
```

図19　FPUのレジスタ




```
        ⋮
        ⋮

COMMON:
    eret
```

まあ，実際にはこんなに単純ではなく，評判の悪い割り込み応答性を少しでもよくするために，ハンドラ内でEXLをクリアしたり，そのためにEPCレジスタの値をメモリに退避/回復したりしているようである。割り込み要因の切り分けもテーブル参照で分岐すれば高速化できる可能性もある（というか，実際にはやってるんだろうな，おそらく）。

コプロセッサ1

MIPS系のMPUは浮動小数点演算をサポートしている。浮動小数点ユニット（FPU）はCPUとは別のコプロセッサ1として実装されている。R3000ではFPUは別チップのR3010として供給されていたが，R4000以降は1チップに内蔵されるようになった。性能向上のため，CPUとFPUは別個のハードウェアであることが多いが，R4200/R4300ではチップ面積を節約するため整数演算の論理を流用して浮動小数点演算機能を実現している。

(1) FPUのレジスタセット

図19にFPUのレジスタセットを示す。FPUのレジスタセットは32本の64ビット長の汎用レジスタと2本の制御レジスタで構成される。制御レジスタはステータスレジスタ（FCR31（図20））とプロセッサのリビジョンレジスタ（FCR0）が定義されているのみである。汎用レジスタ（FGR）へのアクセスは，CPUの汎用レジスタ間と，専用命令，

```
mfc1   (Move from Coprocessor 1)
mtc1   (Move to Coprocessor 1)
dmfc1  (Doubleword Move from Coprocessor 1)
dmtc1  (Doubleword Move to Coprocessor 1)
```

を使用して行える。汎用レジスタはメモリ間とデータを直接ロード/ストアすることも可能である。そのための命令が，

図20　浮動小数点ステータスレジスタ（FCR31）

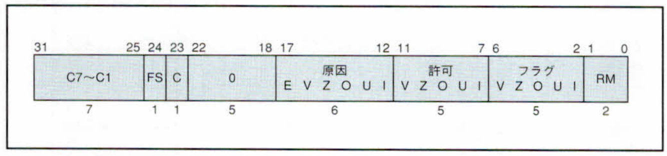


図21　単精度浮動小数点データ形式

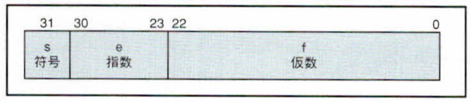
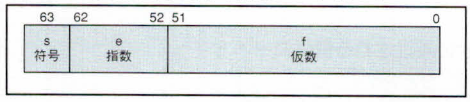


図22　倍精度浮動小数点データ形式




```
lwc1  (Load Word to Coprocessor 1)
ldc1  (Load Doubleword to Coprocessor 1)
swc1  (Store Word from Coprocessor 1)
sdc1  (Store Doubleword from Coprocessor 1)
lwxc1  (Load Word Indexed to Coprocessor 1)              [MIPS-IV]
ldxc1  (Load Doubleword Indexed to Coprocessor 1)        [MIPS-IV]
swxc1  (Store Word Indexed from Coprocessor 1)           [MIPS-IV]
sdxc1  (Sotre Doubleword Indexed from Coprcessor 1)      [MIPS-IV]
```

である。制御レジスタ（FCR）とCPUレジスタ間のデータ転送には，

```
cfc1  (Move Control Word from Coprocessor 1)
ctc1  (Move Control Word to Coprocessor 1)
```

という命令を使用する。

浮動小数点データは単精度（32ビット）と倍精度（64ビット）をサポートする。ステータスレジスタ（図15）のFRビットが0の場合，浮動小数点レジスタ（FPR）は16本である。連続する偶数番号と奇数番号の汎用レジスタ（FGR）の下位32ビットを組にして64ビットの浮動小数点レジスタとみなす。これはCPUの汎用レジスタ長が32ビットだったR3000での仕様の名残で，コプロセッサ1と32ビット単位でしかアクセスできなかったためである。R4000以降も互換性を保っているわけだ。R4000以降は64ビットプロセッサになったため，それに対応して64ビット単位のデータ転送が可能になった。ステータスレジスタのFRビットが1の場合，浮動小数点レジスタは32本となる。これは，カーネルモード，または64ビットモードで使用できる。

浮動小数点レジスタは，単精度または倍精度の浮動小数点データを保持する。FRビットが0の場合は偶数番号のレジスタのみが指定可能である。FRビットが1の場合はすべてのレジスタが使用可能である。

(2) 浮動小数点データの形式

FPUは単精度と倍精度のIEEE754標準の浮動小数点演算をサポートする。単精度の浮動小数点形式は図21に示すように，24ビットの符号付き仮数部（s＋f）と，8ビットの指数部（e）で構成される。倍精度の浮動小数点形式は図22に示すように，53ビットの符号付き仮数部（s＋f）と，11ビットの指数部（e）で構成される。浮動小数点データは，次の3つの領域により成り立っている。

符号ビット：s
指数部　：e＝E＋バイアス値（単精度→0x7f；倍精度→0x3ff）
仮数部　：$f = .b_1b_2b_3b_4\cdots\cdots$（小数点第1位以下の値）

これらの情報で，

$\pm 1.b_1b_2b_3b_4\cdots\cdots \times 2^E$

のデータを表す。浮動小数点のデータ形式では整数部の1を省いて表現している。このような形式で表現できる数を正規化された数（ノーマル数）と呼ぶ。一方，整数部が0である，

$\pm 0.b_1b_2b_3b_4\cdots\cdots \times 2^E$

図23　丸めモード

RM	記号	意味
00	RN	表現可能なもっとも近い値に結果を丸める。もっとも近い値が2つある場合は，最下位ビットが0の値のほうを選ぶ
01	RZ	ゼロの方向に丸める。いわゆる切り捨て
10	RP	＋∞の方向に丸める
11	RM	－∞の方向に丸める

図24　Guard，Round，Stickyビットの意味

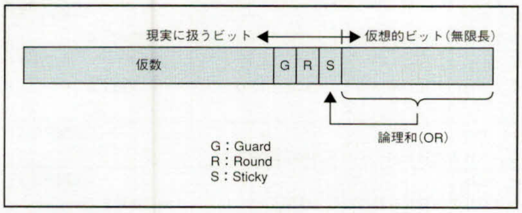


図25　丸めの実現

丸めモード	符号	仮数の最下位	Guard	Round	Sticky	切り上げ切り捨て
RN	×	×	0	×	×	↓
	×	×	1	1	×	↑
	×	×	1	0	1	↑
	×	1	1	0	0	↑
	×	0	1	0	0	↓
RZ	×	×	×	×	×	↓
RP	1	×	×	×	×	↓
	0	×	1	×	×	↑
	0	×	0	1	×	↑
	0	×	0	0	1	↑
	0	×	0	0	0	↓
RM	0	×	×	×	×	↓
	1	×	1	×	×	↑
	1	×	0	1	×	↑
	1	×	0	0	1	↑
	1	×	0	0	0	↓

×はDon't Care

のようなデータ形式を非正規化数（デノーマル数）と呼ぶ。また，IEEE754では非数（NaN），無限大，ゼロというデータを定義している。NaN（Not a Number）とは，指数部（e）が最大値で，仮数部（f）が0以外のデータである。さらに，fの最上位ビット（小数点以下第1位）が1のときをSignalingNaN（SNaN），0のときをQuietNaN（QNaN）と呼ぶ。無限大とは，指数部（e）が最大値で，仮数部（f）が0のデータである。符号（s）によって正の無限大と負の無限大がある。ゼロとは，指数部（e）と仮数部（f）が0のデータである。符号（s）によって正のゼロと負のゼロがある。±1.0，QNaN，±無限大，±ゼロを16進数で表現すると，それぞれ次のようになる。

```
+1.0    0x3f800000            (単精度)
        0x3ff0000000000000    (倍精度)

-1.0    0xbf800000            (単精度)
        0xbff0000000000000    (倍精度)

QNaN    0x7fbfffff            (単精度)
        0x7ff7ffffffffffff    (倍精度)

+∞      0x7f800000            (単精度)
        0x7ff0000000000000    (倍精度)

-∞      0xff800000            (単精度)
        0xfff0000000000000    (倍精度)

+0      0x00000000            (単精度)
        0x0000000000000000    (倍精度)

-0      0x80000000            (単精度)
        0x8000000000000000    (倍精度)
```

（3）丸めモード

FPUは浮動小数点演算の結果をステータスレジスタ（FCR31）のRMビットに応じて丸め（切り上げ，切り捨て）を行う。丸めモードの意味を図23に示す。

FPU内で丸めを実現するために，浮動小数点データに丸め用の補助ビットを3ビット追加して演算を行う。それがGuard，Round，Stickyビットである。GuardとRoundについては単に演算精度を増やすだけであるが，Stickyには特別な意味がある。すなわち，演算の途中で右シフトを行った場合，シフトアウトされるあふれビットのすべての論理和がStickyになる。つまり，IEEE754標準においては，浮動小数点の仮数部を扱うとき，あたかも無限の精度があるように扱う。物理的には3ビットしか余分なビットがないが，Sticky以下の重みの（右にある）ビットは，すべてStickyビットに代表される。すなわち図24のようなイメージである。

Guard，Round，Stickyビットは，演算前の仮数を3ビット左シフトして生成する。したがって，これらのビットの初期値は0であり，桁あわせなどで右シフトが発生するときに変化する（上位の値が繰り下がってくる。ただしStickyビットはシフトアウトされる値との論理和）。丸めは，結果の符合，仮数の最下位ビット，Guard，Round，Stickyビットの値と丸めモードとの関係で行われる。具体的には図25に示すような関係を見て，仮数の切り上げ，切り捨てを決定する。

ところで，なにかと話題のPlayStation2のMPUであるEmotionEngineはGuardビットのみで丸めを行っているようである。最初はゼロ方向への丸め（RZ）のみでいきたかったと推測される（その場合，Guard，Round，Stickyビットは不要）が，3次元グラフィック処理では精度が不足したのであろう。ちょっと中途半端である。

（3）浮動小数点例外

FPUはIEEE754の定める，次の5種類の例外を，ステータスレジスタの，原因ビット，許可ビット，フラグビットによりサポートしている。

- 不正演算（I）（精度落ちのこと）
- オーバーフロー（O）
- アンダーフロー（U）
- ゼロ除算（Z）
- 無効演算（V）

このほかに，FPUでは6番目の例外として，未実装演算（E）例外を定義している。これは，浮動小数点演算を実行できないとき，例外事象を正常に処理できないときに発生する。未実装例外には許可ビットはなく，必ず例外が発生する。

FPUは演算処理で例外が発生すると，対応する原因ビットに例外要因をセットする。このとき，対応する許可ビットがセットされていれば，浮動小数点例外を発生する。もし，許可ビットがセットされていなければ，対応するフラグビットの値と論理和をとる。フラグビットには処理されなかった（例外にならなかった）例外要因が蓄積されている。ある処理単位では例外の発生をマスクしておき，その処理中に発生した例外をあとでまとめて処理する場合に参照する。図26にステータスレジスタの各ビットの対応関係を示す。

（4）浮動小数点命令のレパートリー

図27にFPUのサポートする命令の一覧を示す。浮動小数点演算はコプロセッサ0のステータスレジスタのCU1ビットがセットされているときのみ実

図26　FCR31の原因/許可/フラグビット

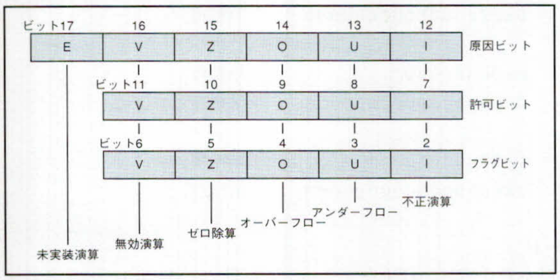


図27-1　FPUロード/ストア命令

命令	形式と説明　［op / base / ft / offset］
Load Word to Floating Point	LWC1　ft,offset (base)　(MIPS Ⅰ) 符号拡張したoffsetをCPUレジスタbaseの内容に加算しアドレスを生成する アドレスから32ビットデータをリードしFPUレジスタftに格納する
Load Double-Word to Floating Point	LDC1　ft,offset (base)　(MIPS Ⅱ) 符号拡張したoffsetをCPUレジスタbaseの内容に加算しアドレスを生成する アドレスから64ビットデータをリードしFPUレジスタftに格納する
Store Word to Floating Point	SWC1　ft,offset (base)　(MIPS Ⅰ) 符号拡張したoffsetをCPUレジスタbaseの内容に加算しアドレスを生成する FPUレジスタftの32ビットデータをアドレスにライトする
Store Double-Word to Floating Point	SDC1　ft,offset (base)　(MIPS Ⅱ) 符号拡張したoffsetをCPUレジスタbaseの内容に加算しアドレスを生成する FPUレジスタftの64ビットデータをアドレスにライトする

命令	形式と説明　［COP1X / base / index / fs / fd / func］
Load Word Indexed to Floating Point	LWXC1　fd,index (base)　(MIPS Ⅳ) CPUレジスタindexとbaseの内容を加算しアドレスを生成する アドレスから32ビットデータをリードしFPUレジスタfdに格納する
Load Double-Word Indexed to Floating Point	LDXC1　fd,index (base)　(MIPS Ⅳ) CPUレジスタindexとbaseの内容を加算しアドレスを生成する アドレスから64ビットデータをリードしFPUレジスタfdに格納する
Store Word Indexed to Floating Point	SWXC1　fs,index (base)　(MIPS Ⅳ) CPUレジスタindexとbaseの内容を加算しアドレスを生成する FPUレジスタfsの32ビットデータをアドレスにライトする
Store Double-Word Indexed to Floating	SDXC1　fs,index (base)　(MIPS Ⅳ) CPUレジスタindexとbaseの内容を加算しアドレスを生成する FPUレジスタfsの64ビットデータをアドレスにライトする

命令	形式と説明　［COP1X / base / index / hint / 0 / func］
Prefetch Indexed	PREFX　hint,index (base)　(MIPS Ⅳ) CPUレジスタindexとbaseの内容を加算しアドレスを生成する hintが示す情報に従い，アドレスで指定されたデータをキャッシュにプリフェッチする

図27-2　FPU演算命令

命令	形式と説明　［COP1 / fmt / ft/cc / fs / fd/cc / func］
Add	ADD.fmt　fd,fs,ft　(MIPS Ⅰ) FPUレジスタfsとftの内容を加算し結果をFPUレジスタfdに格納する
Subtract	SUB.fmt　fd,fs,ft　(MIPS Ⅰ) FPUレジスタfsの内容からftの内容を減算し結果をFPUレジスタfdに格納する
Multiply	MUL.fmt　fd,fs,ft　(MIPS Ⅰ) FPUレジスタfsとftの内容を乗算し結果をFPUレジスタfdに格納する
Divide	DIV.fmt　fd,fs,ft　(MIPS Ⅰ) FPUレジスタfsの内容をftの内容で除算し結果をFPUレジスタfdに格納する
Square Root	SQRT.fmt　fd,fs　(MIPS Ⅱ) FPUレジスタfsの内容の平方根を求め結果をFPUレジスタfdに格納する
Absolute Value	ABS.fmt　fd,fs　(MIPS Ⅰ) FPUレジスタfsの内容の絶対値を求め結果をFPUレジスタfdに格納する
Move	MOV.fmt　fd,fs　(MIPS Ⅰ) FPUレジスタfsの内容をFPUレジスタfdに格納する
Negate	NEG.fmt　fd,fs　(MIPS Ⅰ) FPUレジスタfsの内容を符号反転して結果をFPUレジスタfdに格納する
※1	MOVT.fmt　fd,fs,cc　(MIPS Ⅳ) 浮動小数点条件ccが1なら，FPUレジスタfsの内容をFPUレジスタfdに格納する
※2	MOVF.fmt　fd,fs,cc　(MIPS Ⅳ) 浮動小数点条件ccが0なら，FPUレジスタfsの内容をFPUレジスタfdに格納する
※3	MOVZ.fmt　fd,fs,rt　(MIPS Ⅳ) CPUレジスタrtの値が0なら，FPUレジスタfsの内容をFPUレジスタfdに格納する
※4	MOVN.fmt　fd,fs,rt　(MIPS Ⅳ) CPUレジスタrtの値が0以外なら，FPUレジスタfsの内容をFPUレジスタfdに格納する
※5	RECIP.fmt　fd,fs　(MIPS Ⅳ) FPUレジスタfsの逆数をFPUレジスタfdに格納する
※6	RSQRT.fmt　fd,fs　(MIPS Ⅳ) FPUレジスタfsの平方根を計算し，その逆数をFPUレジスタfdに格納する
※7	ROUND.L.fmt　fd,fs　(MIPS Ⅲ) FPUレジスタfsの内容をもっとも近い64ビット整数に変換し，結果をfdに格納する
※8	TRUNC.L.fmt　fd,fs　(MIPS Ⅲ) FPUレジスタfsの内容を小数点以下を無視して64ビット整数に変換し，結果をfdに格納する
※9	CEIL.L.fmt　fd,fs　(MIPS Ⅲ) FPUレジスタfsの内容を+∞方向に丸め，64ビット整数に変換し，結果をfdに格納する
※10	FLOOR.L.fmt　fd,fs　(MIPS Ⅲ) FPUレジスタfsの内容を-∞方向に丸め，64ビット整数に変換し，結果をfdに格納する
※11	ROUND.W.fmt　fd,fs　(MIPS Ⅱ) FPUレジスタfsの内容をもっとも近い32ビット整数に変換し，結果をfdに格納する
※12	TRUNC.W.fmt　fd,fs　(MIPS Ⅱ) FPUレジスタfsの内容を小数点以下を無視して32ビット整数に変換し，結果をfdに格納する
※13	CEIL.W.fmt　fd,fs　(MIPS Ⅱ) FPUレジスタfsの内容を+∞方向に丸め，32ビット整数に変換し，結果をfdに格納する
※14	FLOOR.W.fmt　fd,fs　(MIPS Ⅱ) FPUレジスタfsの内容を-∞方向に丸め，32ビット整数に変換し，結果をfdに格納する
※15	CVT.S.fmt　fd,fs　(MIPS Ⅰ,Ⅲ) FPUレジスタfsの内容を単精度浮動小数点データに変換し，結果をfdに格納する
※16	CVT.D.fmt　fd,fs　(MIPS Ⅰ,Ⅲ) FPUレジスタfsの内容を倍精度浮動小数点データに変換し，結果をfdに格納する
※17	CVT.L.fmt　fd,fs　(MIPS Ⅲ) FPUレジスタfsの内容を64ビット整数データに変換し，結果をfdに格納する
※18	CVT.W.fmt　fd,fs　(MIPS Ⅰ) FPUレジスタfsの内容を32ビット整数データに変換し，結果をfdに格納する
Compare	C.cond.fmt　fs,ft　(MIPS Ⅰ) FPUレジスタfsの内容とFPUレジスタftの内容を比較し結果をcc0に設定する C.cond.fmt　cc,fs,ft　(MIPS Ⅳ) FPUレジスタfsの内容とFPUレジスタftの内容を比較し結果を指定されたccに設定する

行可能である。MIPS IVの命令に関しては，CU3ビット(XXビットとかMIPS 4ビットとか呼ばれる)もセットされている必要がある。

図28に一般的なFPU演算のデータパスを示す。浮動小数点データは，まず，符号(s)，指数(e)，仮数(f)に分割され，仮数には，Guard，Round，Stickyビットが付加される。そのあとは，符号，指数，仮数ごとに別々に計算が行われる。最後に，仮数の演算結果を丸めモード(RM)に従って丸め，符号，指数，仮数を再結合して結果を得る。MIPS系のMPUの多くは丸め後の指数の値を調べて，オーバーフロー，アンダーフローを検出する。

図27-2　FPU演算命令（つづき）

命令	形式と説明
	COP1X | fr | ft | fs | fd | func
Multiply Add	MADD.fmt　fd,fr,fs,ft　（MIPS Ⅳ） FPUレジスタfsとftの内容を乗算し，結果にFPUレジスタfrの内容を加算し，結果をFPUレジスタfdに格納する
Multiply Subtract	MSUB.fmt　fd,fr,fs,ft　（MIPS Ⅳ） FPUレジスタfsとftの内容を乗算し，結果からFPUレジスタfrの内容を減算し，結果をFPUレジスタfdに格納する
Negative Multiply Add	NMADD.fmt　fd,fr,fs,ft　（MIPS Ⅳ） FPUレジスタfsとftの内容を乗算し，結果にFPUレジスタfrの内容を加算し，符号反転の結果をFPUレジスタfdに格納する
Negative Multiply Subtract	NMSUB.fmt　fd,fr,fs,ft　（MIPS Ⅳ） FPUレジスタfsとftの内容を乗算し，結果からFPUレジスタfrの内容を減算し，符号反転の結果をFPUレジスタfdに格納する

※1　Move Conditional on Floating Point True
※2　Move Conditional on Floating Point False
※3　Move Conditional on Zero
※4　Move Conditional on Not Zero
※5　Reciprocal Approximation
※6　Reciprocal Square Root Approximation
※7　Round to Long Fixed Point
※8　Truncate to Long Fixed Point
※9　Ceiling Convert to Long Fixed Point
※10　Flooring Convert to Long Fixed Point
※11　Round to Word Fixed Point
※12　Truncate to Word Fixed Point
※13　Ceiling Convert to Word Fixed Point
※14　Flooring Convert to Word Fixed Point
※15　Convert to Single Floating Point
※16　Convert to Double Floating Point
※17　Convert to Long Fixed Point
※18　Convert to Word Fixed Point

図27-3　FPU分岐命令

命令	形式と説明
	COP1 | BC1 | cc | offset
Branch on FP True	BC1T　offset　（MIPS Ⅰ） 浮動小数点条件ビットcc0が1なら1命令遅れて分岐先へ分岐する
	BC1T　cc,offset　（MIPS Ⅳ） 指定された浮動小数点条件ビットccが1なら1命令遅れて分岐先へ分岐する
Branch on FP True Likely	BC1TL　offset　（MIPS Ⅰ） 浮動小数点条件ビットcc0が1なら1命令遅れて分岐先へ分岐する。cc0が0なら遅延スロットを実行しない
	BC1TL　cc,offset　（MIPS Ⅳ） 指定された浮動小数点条件ビットccが1なら1命令遅れて分岐先へ分岐する。ccが0なら遅延スロットを実行しない
Branch on FP False	BC1F　offset　（MIPS Ⅰ） 浮動小数点条件ビットcc0が0なら1命令遅れて分岐先へ分岐する
	BC1F　cc,offset　（MIPS Ⅳ） 指定された浮動小数点条件ビットccが0なら1命令遅れて分岐先へ分岐する
Branch on FP False Likely	BC1FL　offset　（MIPS Ⅰ） 浮動小数点条件ビットcc0が0なら1命令遅れて分岐先へ分岐する。cc0が1なら遅延スロットを実行しない
	BC1FL　cc,offset　（MIPS Ⅳ） 指定された浮動小数点条件ビットccが0なら1命令遅れて分岐先へ分岐する。ccが1なら遅延スロットを実行しない

図29に各MPUでの浮動小数点演算の実行クロック数を示す。R3000の時代（外付けのFPUであるR3010で実行）では2クロックで実行できていた加算（ADD）が，R4000，R5000と動作周波数が向上していくにつれて実行に多くのクロック数がかかるようになっている。これはMPU内部の演算器の動作周波数（処理時間）が，MPUの動作周波数に追いついていかないためで，浮動小数点演算の処理性能（処理時間）自体は格段に向上してきている。ちなみに，R3010は40MHz動作（1クロックは25n秒），R5000は200MHz動作（1クロックは5n秒）なので，単位クロックの時間は5倍高速になっている。また，演算処理は，

　符号/指数/仮数の分離→演算→丸め→符号/指数/仮数の結合

というように，段階を分けてパイプライン処理されるので，浮動小数点演算が連続する場合には，外見えの処理クロック数は1～3クロック程度短くなる。

図27-4　FPU転送命令

命令	形式と説明
	COP1 | func | rt | fs | 0
Move Word From Floating Point	MFC1　rt,fs　（MIPS Ⅰ） 32ビットデータをFPUレジスタfsからCPUレジスタrtに転送する
Move Word To Floating Point	MTC1　rt,fs　（MIPS Ⅰ） 32ビットデータをCPUレジスタrtからFPUレジスタfsに転送する
Move Double-Word From Floating Point	DMFC1　rt,fs　（MIPS Ⅲ） 64ビットデータをFPUレジスタfsからCPUレジスタrtに転送する
Move Double-Word To Floating Point	DMTC1　rt,fs　（MIPS Ⅲ） 64ビットデータをCPUレジスタrtからFPUレジスタfsに転送する
Move Control Word From Floating Point	CFC1　rt,fs　（MIPS Ⅲ） FPU制御レジスタfsの内容をCPUレジスタrtに転送する
Move Control Word To Floating Point	CTC1　rt,fs　（MIPS Ⅲ） CPUレジスタrtの内容をFPU制御レジスタfsに転送する

（5）演算精度に関して

IEEE754標準では，浮動小数点演算は，浮動小数点データが無限の精度を持っているように行われる。このため中間の演算の精度をどのような精度で行おうとも結果は同じにならなければならない。たとえば，単精度の浮動小数点演算を，そのままの精度で実行しても，倍精度で実行しても，あるいは内部的に80ビットで実行しても，結果を丸めて単精度にした場合はすべて同じ結果にならなければならない。これは，Guard，Round，Stickyビット（特にStickyビット）を付加することで実現できるとされている。ただし，積和演算のように，乗算のあとに加算という2回以上の演算を実行するときは，実行速度を上げるために，1回目の演算結果を丸めずに，そのまま2回目の演算を行う場合がある。このような場合は，内部の演算精度によって結果が異なることがある。MIPSの命令セットでいえば，

```
madd    (Multiply Add)
msub    (Multiply Subtract)
nmadd   (Multiply Add and Negate)
nmsub   (Multiply Subtract and Negate)
rsqrt   (Reciprocal Square Root Approximation)
```

がそのような命令にあたる。また，

```
recip   (Reciprocal Approximation)
```

については，データを1．0から除算するだけなので，どのように計算しても同じ結果になるはずであるが，仕様では結果の一致性を保証していない。これは逆数の高速近似アルゴリズムで演算を実行する場合を考慮してのものであろう。同じことはrsqrt命令についてもいえる。

R3000系との違い

（1）例外/割り込み処理

R4000系MPUとR3000系MPUの大きな違いは例外/割り込み処理にあ

図28　FPVの一般的なデータパス

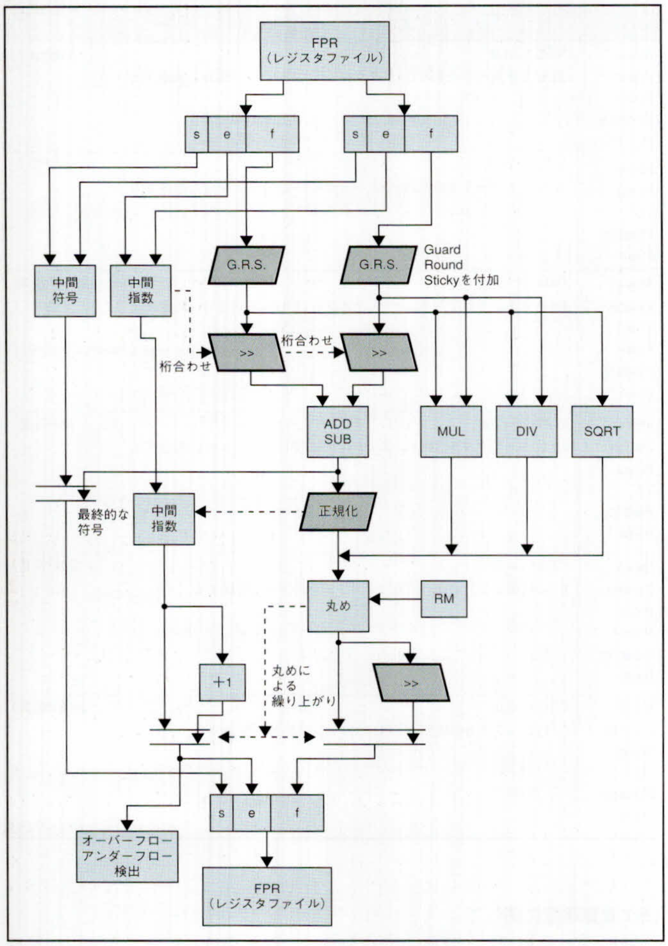
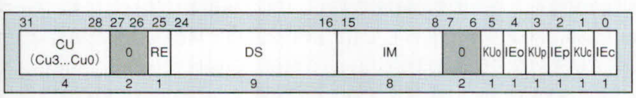


図29　FPU命令の実行クロック数

命令	R3010	R4000	R4200	R4300	R5000	R5400
ADD.S	2	4	3	3	4	4
ADD.D	2	4	3	−3	4	4
SUB.S	2	4	3	3	4	4
SUB.D	2	4	3	3	4	4
MUL.S	4	7	11	5	4	5
MUL.D	5	8	20	8	5	6
MADD.S	N/A	N/A	N/A	N/A	4	9
MADD.D	N/A	N/A	N/A	N/A	5	10
DIV.S	12	23	29	29	21	31
DIV.D	19	36	58	58	36	59
SQRT.S	N/A	54	30	29	21	31
SQRT.D	N/A	112	59	58	36	59
RECIP.S	N/A	N/A	N/A	N/A	21	31
RECIP.D	N/A	N/A	N/A	N/A	36	59
RSQRT.S	N/A	N/A	N/A	N/A	21	61
RSQRT.D	N/A	N/A	N/A	N/A	36	121
ABS.S	1	2	1	1	1	2
ABS.D	1	2	1	1	1	2
MOV.S	1	1	1	1	1	2
MOV.D	1	1	1	1	1	2
NEG.S	1	2	1	1	1	2
NEG.D	1	2	1	1	1	2
ROUND.W	N/A	4	5	5	4	6
ROUND.L	N/A	4	5	5	4	6
CVT.D	1	2	5	5	4	6
CVT.S	2	4	5	5	4	6
C.cnd.S	2	3	1	1	−1	2
C.cnd.D	2	3	1	1	1	2

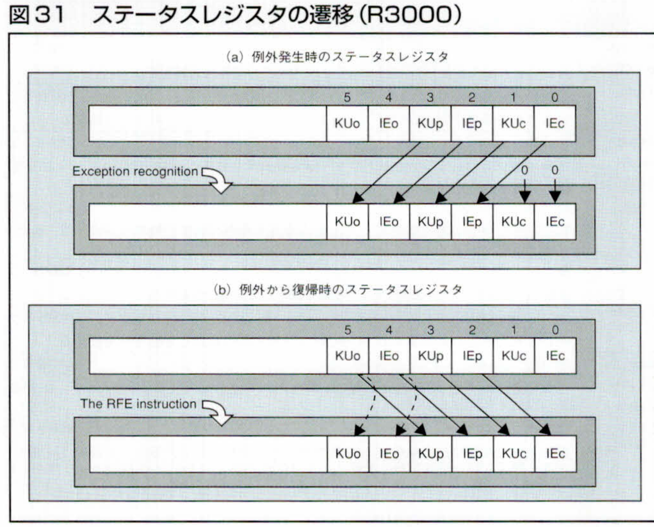
(N/A：Not Available)

図30　ステータスレジスタ(R3000)



CU　：コプロセッサの使用許可指定(1→使用可，0→使用不可)．
　　　カーネルモードではCU0の値にかかわらずCP0は使用可能．
RE　：ユーザーモードでのエンディアンの反転指定．
DS　：自己診断領域．
IM　：割り込みマスク．
KUo：古いカーネル/ユーザーモード(0→カーネル，1→ユーザー)．
IEo　：古い割り込み許可(0→禁止，1→許可)．
KUp：ひとつ前のカーネル/ユーザーモード(0→カーネル，1→ユーザー)．
IEp　：ひとつ前の割り込み許可(0→禁止，1→許可)．
KUc：現在のカーネル/ユーザーモード(0→カーネル，1→ユーザー)．
IEc　：現在の割り込み許可(0→禁止，1→許可)．

図31　ステータスレジスタの遷移(R3000)



る。図30にR3000のステータスレジスタ(コプロセッサ0)を示す。図30でわかるように，R3000の保護レベルはカーネルとユーザーの2種類である。例外処理が開始されるとMPUはカーネルモード(に相当するレベル)で割り込み不可状態に移行する。この操作をR4000系以降のMPUではステータスレジスタのEXLビットをセットすることにより実現する。R3000では，保護レベル(KUx)と割り込み許可状態(IEx)を左シフトすることで，前の状態を保存し，新たな保護レベルと割り込み許可状態に0を設定(カーネルで割り込み不可)する。例外処理から復帰する場合は，

```
rfe  (Restore from Exception)
```

という命令で，保護レベルと割り込み許可状態を右シフトして前の状態を回復する。この状況を図31に示す。rfe命令は，eret命令とは異なり，自動的に例外で中断した命令への分岐は行わない。

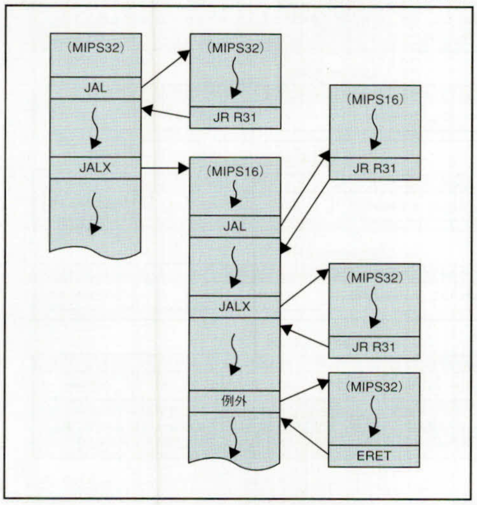
```
mfc0    r1,EPC
jr      r1
rfe
```

のように，復帰するためのジャンプ命令の遅延スロットにrfe命令を配置し，パイプラインを回復する。R4000系では例外発生時に，EXLで示される「例外レベル」とERLで示される「エラーレベル」のどちらかに移行する。このとき戻りアドレスが格納されるレジスタがEPCとエラーEPCに分かれる。R3000のような方法を採用すると，現在のレベルが「例外レベル」か「エラーレベル」であるかを識別し，どちらのレジスタから戻りアドレスを取り出してジャンプするのか決定しなければならない。この手間を省くためにeret命

図32　MIPS16のレジスタ

MIPS16での エンコード	MIPS32での エンコード	記号	説明
0	16	s0	汎用レジスタ
1	17	s1	汎用レジスタ
2	2	v0	汎用レジスタ
3	3	v1	汎用レジスタ
4	4	a0	汎用レジスタ
5	5	a1	汎用レジスタ
6	6	a2	汎用レジスタ
7	7	a3	汎用レジスタ
—	24	t8	条件コードレジスタ。BTEQZ, BTNEZ,CMP, CMPI, SLT, SLTU, SLTI,SLTIU命令で暗黙的に参照される
—	29	sp	スタックポインタ
—	31	ra	リターンアドレスレジスタ
—	—	PC	プログラムカウンタ
—	—	HI	乗除算結果の格納用。上位，剰余
—	—	LO	乗除算結果の格納用。下位，商

図33　MIPS16とMIPS32間のモード遷移



令が導入されたのであろう（と昔に考えたことがあったような）。

（2）MMU

R3000のMMUは，ページサイズが4Kバイト固定で，ひとつの仮想アドレスをひとつの物理アドレスに対応させる。R4000系以降で採用されたダブルエントリではない。しかし，最近のR3000をベースにした組み込み制御用のMPUでも，ダブルエントリ方式のTLBであることが多い。例外/割り込み処理においても，rfe方式ではなく，eret方式を採用するほうが多い。つまり，32ビットモードのみのR4000という位置付けである。R3000方式のMMUはいまや忘れ去られようとしている。

MIPS16

（1）概要

MIPS16命令セットはARM社が自社の命令セットを拡張して16ビットの命令長に対応させたThumb命令セットに対抗するものとしてLSIロジックによって提唱された。その目的は，組み込み制御分野におけるプログラムのメモリ占有率を低減することである。それをMIPS社が自社のアーキテクチャの拡張として認定したかたちになっている。基本命令長は16ビットで，32ビット長のMIPS本来の命令セット（以下ではMIPS32と呼ぶ）と混在して使用できる。Lexra社の試算によると，命令コードの容量はMIPS32に比べて60%の大きさになるという。MIPS32の命令が32ビット長なので50%ではないかと思う人がいるかもしれない。現実には16ビット命令だけではなく，32ビット長命令との混合になるのでまったく半分のコードサイズにはならない。ともあれ，MIPS16はパームサイズ用のWindowsCEなどでの活用が期待されている。MIPS16の特徴を箇条書きにすると次のようになる。

- 16ビット長の命令形式
- MIPS32と混在可能
- 8/16/32/64ビット長のデータをサポート
- 8本の汎用レジスタと，いくつかの特殊レジスタを使用可能
- 特権命令は定義されていない（必要な場合はMIPS32で行う）

MIPS16をサポートしているMPUは結構存在する。本家であるLSIロジックのTR4101/4102，Lexra社のLX4180，東芝のTX19，NECのVR4111/4121/4122などである。NECのVR41xxシリーズ以外は32ビットモードのみのサポートであり，64ビット命令と64ビットデータを使用することはできない。最近MIPSから発表されたOPAL（5Kc）もMIPS16のサポートを匂わせているが，こちらはどうなるか不明である。

（2）レジスタセット

図32にMIPS16のレジスタセットを示す。これらのレジスタはMIPS32のレジスタの一部分であり，MIPS16からMIPS32，あるいはMIPS32からMIPS16へのモード遷移で保存される。MIPS16における汎用レジスタは，MIPS32のうちから8本が使用できる。単純にはR0からR7が割り当てられるところであるが，MIPS32において，R0は値が0固定，R1はプログラムのテンポラリレジスタであるので，それらの割り当てを避け，代わりにR16とR17を利用する。これら8本の汎用レジスタに加え，命令によっては，スタックポインタ（SP，R29），リターンアドレスレジスタ（RA，R31），条件コードレジスタ（R24），プログラムカウンタ（PC）を参照できる。

また，MOVE命令を使用すれば，MIPS32のすべての汎用レジスタとの間でデータの転送が可能である。MIPS16では，汎用レジスタが8本であるため，実際のプログラムではスタック領域へのレジスタの退避と回復が頻繁に出現する。メモリアクセスが増加すればプログラムの処理性能は低下するが，MOVE命令を使用してMIPS32のレジスタを退避領域に用いればメモリアクセスをなくすことができる。

（3）モード遷移

MPUはPCの最下位ビットが1のとき，MIPS16モードで命令を実行する。したがって，PCの最下位ビットをセット/リセットすることで，MIPS16とMIPS32間でモード遷移を行える。これを実現する方法には次のようなものがある。

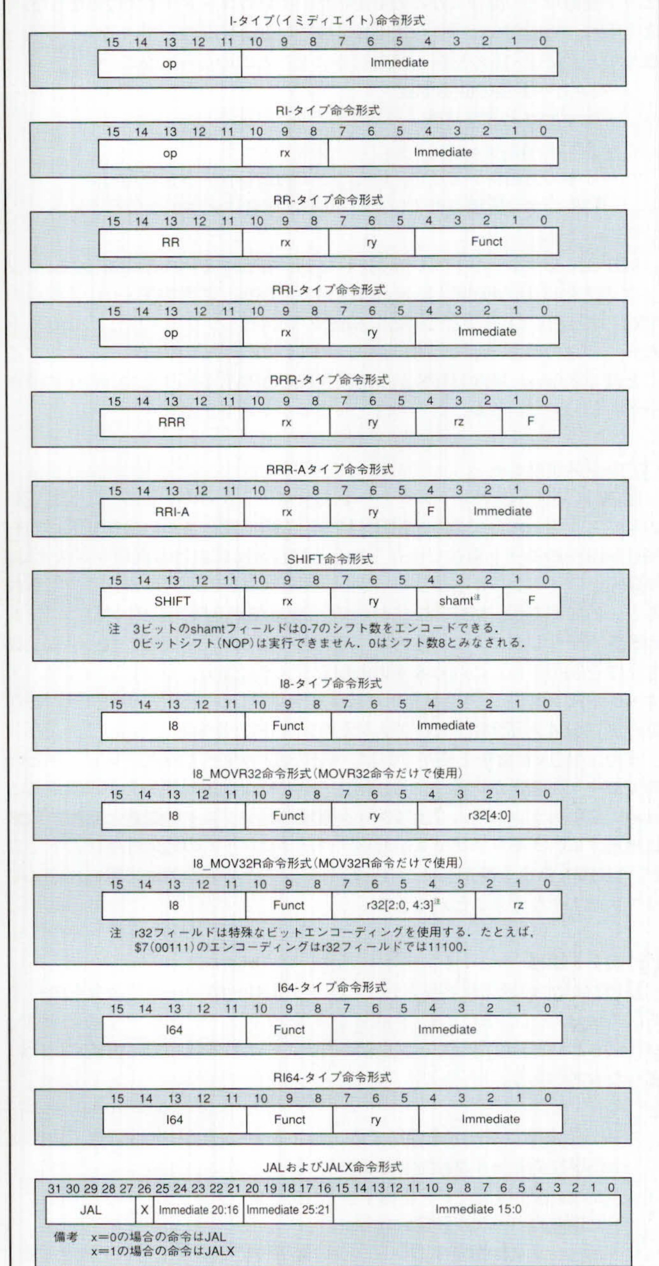
- JAL命令はモードを保持する。RA（リターンレジスタ）の最下位ビットに現在のモードが設定される。
- JALX命令はモードを逆転する。RA（リターンレジスタ）の最下位ビットに現在のモードが設定される。
- JALR命令は指定するレジスタの最下位ビットがモードを決定する。RA（リターンレジスタ）の最下位ビットに現在のモードが設定される。
- JR命令は指定するレジスタの最下位ビットがモードを決定する。
- ERET命令（MIPS32のみ）はEPCの最下位ビットがモードを決定する。
- 例外/割り込みが発生すると強制的にMIPS32モードになる。

各モード遷移のイメージを図33に示す。

（4）MIPS16の命令形式

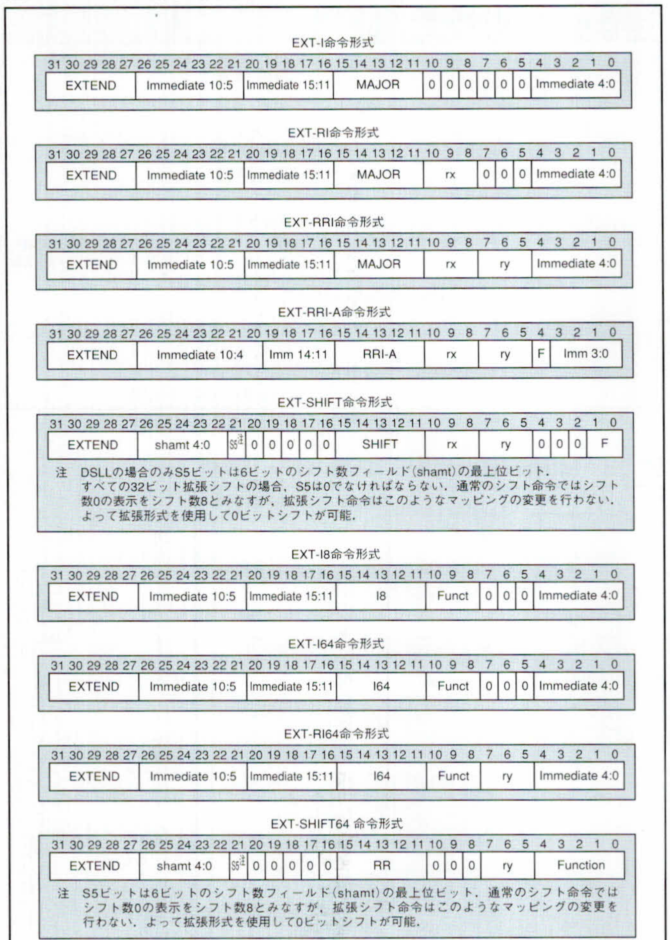
MIPS16の命令は16ビット長を持ち，ハーフワード（16ビット）境界に配置される。図34にMIPS16の命令形式を示す。種類が多いようにも思えるが，基本的には2オペランド形式（MIPS32は3オペランド形式）で，16ビットのうち，5ビットが命令の種類，3ビット2つが各レジスタ番号，残る5ビットがイミディエイト値を示す。JALとJALXについては，16ビット長では分岐先を指定できないので，32ビット長となっている。ところで，イミディエイト値が5ビットでは，0から31までの値しか指定できないので，プ

図34　16の命令形式(EXTENDなし)



フィールド	定義
op	5ビットのメジャーオペコード
rx	3ビットのソース/デスティネーションレジスタ指定
ry	3ビットのソース/デスティネーションレジスタ指定
immediate またはimm	4ビット，5ビット，8ビット，または11ビットのイミディエイト値，分岐ディスプレースメント，またはアドレスディスプレースメント
rz	3ビットのソース/デスティネーションレジスタ指定
FunctまたはF	機能フィールド

図35　16の命令形式(EXTEND付き)



ログラムを書く上で不便である。そこで，EXTENDプリフィックスが用意されている。イミディエイト領域を持つ命令はEXTENDプリフィクスを付けることで32ビット長に拡張できる。この場合，イミディエイト値は16ビットの範囲まで指定できる。図35にEXTENDされた命令形式を示す。

(5) MIPS16の命令セット

MIPS16の命令セットの詳細を図36-1から36-7に示す。また，オペコードマップを図37に示す。図を見ればわかるが，MIPS16の命令は，ニーモニック的には，MIPS32の命令のサブセットである。MIPS32と同じく，ロード/ストアアーキテクチャを採用している。MIPS32に慣れ親しんでいる人には馴染みやすいであろう。違いは，MIPS32が3オペランド形式を標準としているのに対し，MIPS16は2オペランド形式であることだ。とはいえ，必要性の高い加減算命令は3オペランド形式を維持している。

MIPS32の命令セットとの最大の違いは，分岐命令に遅延スロットがないことである。RISCのパイプラインの性格上，分岐命令が分岐先の命令を実行するまでに，1～2クロックのアイドル期間が生じる。この期間に分岐命令の後続1命令を実行して，パイプライン処理の無駄をなくすというのが遅延分岐の考え方である。MIPS16の分岐命令においても，同様にアイドル期間は存在するが，分岐が成立する場合，遅延スロットの命令は抹殺される。なぜ，このような無駄を生じる仕様にしているのか，LSIロジックに聞いてみなければ真相はわからない。おそらく，遅延スロットがないほうがコンパイラでの最適化が容易であるためだとは思うのだが。そのくせ，ジャンプ命令にはしっかりと遅延スロットが存在する。

また，PC相対アドレッシングを使用する命令はMIPS32にはないものだ。これはなくても特に不自由はしないのだが，命令コード中に埋め込まれた定数値やテーブルへのアクセスを簡潔に実現するために導入されたのであろう(コンパイラではSWITCH文の実現に利用できる)。その証拠に，ロード命

図36-1　ロード/ストア命令

命令	形式
Load Byte	LB　ry,offset(rx)
Load Byte Unsigned	LBU　ry,offset(rx)
Load Halfword	LH　ry,offset(rx)
Load Halfword Unsigned	LHU　ry,offset(rx)
Load Word	LW　ry,offset(rx)
	LW　rx,offset(pc)
	LW　rx,offset(sp)
Load Word Unsigned	LWU　ry,offset(rx)
Load Doubleword	LD　ry,offset(rx)
	LD　ry,offset(pc)
	LD　ry,offset(sp)
Store Byte	SB　ry,offset(rx)
Store Halfword	SH　ry,offset(rx)
Store Word	SW　ry,offset(rx)
	SW　rx,offset(sp)
	SW　ra,offset(sp)
Store Doubleword	SD　ry,offset(rx)
	SD　ry,offset(sp)
	SD　ra,offset(sp)

図36-2　ALUイミディエイト命令

命令	形式
Load Immediate	LI　rx,immediate
Add Immediate Unsigned	ADDIU ry,rx,immediate
	ADDIU rx,immediate
	ADDIU sp,immediate
	ADDIU rx,pc,immediate
	ADDIU rx,sp,immediate
Doubleword Add Immediate Unsigned	DADDIU ry,rx,immediate
	DADDIU ry,immediate
	DADDIU sp,immediate
	DADDIU ry,pc,immediate
	DADDIU ry,sp,immediate
Set on Less Than Immediate	SLTI　rx,immediate
Set on Less Than Immediate Unsigned	SLTIU　rx,immediate
Compare Immediate	CMPI　rx,immediate

図36-3　2/3オペランド命令

命令	形式
Add Unsigned	ADDU　rz,rx,ry
Subtract Unsigned	SUBU　rz,rx,ry
Doubleword Add Unsigned	DADDU　rz,rx,ry
Doubleword Subtract Unsigned	DSUBU　rz,rx,ry
Set on Less Than	SLT　rx,ry
Set on Less Than Unsigned	SLTU　rx,ry
Compare	CMP　rx,ry
Negate	NEG　rx,ry
AND	AND　rx,ry
OR	OR　rx,ry
Exclusive OR	XOR　rx,ry
NOT	NOT　rx,ry
Move	MOVE　ry,r32

図36-4　シフト命令

命令	形式
Shift Left Logical	SLL　rx,ry,immediate
Shift Right Logical	SRL　rx,ry,immediate
Shift Right Arithmetic	SRA　rx,ry,immediate
Shift Left Logical Variable	SLLV　ry,rx
Shift Right Logical Variable	SRLV　ry,rx
Shift Right Arithmetic Variable	SRAV　ry,rx
Doubleword Shift Left Logical	DSLL　rx,ry,immediate
Doubleword Shift Right Logical	DSRL　ry,immediate
Doubleword Shift Right Arithmetic	DSRA　ry,immediate
Doubleword Shift Left Logical Variable	DSLLV　ry,rx
Doubleword Shift Right Logical Variable	DSRLV　ry,rx
Doubleword Shift Right Arithmetic Variable	DSRAV　ry,rx

図36-5　乗除算命令

命令	形式
Multiply	MULT　rx,ry
Multiply Unsigned	MULTU　rx,ry
Divide	DIV　rx,ry
Divide Unsigned	DIVU　rx,ry
Move From HI	MFHI　rx
Move From LO	MFLO　rx
Doubleword Multiply	DMULT　rx,ry
Doubleword Multiply Unsigned	DMULTU rx,ry
Doubleword Divide	DDIV　rx,ry
Doubleword Divide Unsigned	DDIVU　rx,ry

図36-6　ジャンプ/分岐命令

命令	形式
Jump And Link	JAL　target
Jump And Link Exchange	JALX　target
Jump Register	JR　rx
	JR　ra
Jump And Link Register	JALR　rx
Branch on Equal to Zero	BEQZ　rx,immediate
Branch on Not Equal to Zero	BNEZ　rx,immediate
Branch on T Equal to Zero	BTEQZ　rx,immediate
Branch on T Not Equal to Zero	BTNEZ　rx,immediate
Branch Unconditional	B　immediate

図36-7　特殊命令

命令	形式
Breakpoint	BREAK immediate
Extend	EXTEND immediate

令にはPC相対アドレッシングがあるのに，ストア命令にはない。読み込み専用を仮定しているのは明らかである。このほかにも，コンパイラでの使用を仮定したと思われる命令がいくつかある。まず，加算命令が符合なしのみである。MIPSアーキテクチャでは符合つき加算の実行でオーバーフローが発生すると例外を発生する。これはC言語の仕様とは相容れないので，コンパイラは使用しない。また，RA(リターンアドレス)レジスタの特別扱いや，スタックポインタ相対アドレッシングなどもコンパイラ用と思われる。

(6)MIPS16の実装

MIPS16の命令セットはMIPS32のサブセットであり，MIPS32の命令と1対1に対応させることができる。MIPS16をMPUに実装する場合，この特徴を利用できる。すなわち，MIPS16の命令コードをいったんMIPS32の命令コードに変換してから，MIPS32用のデータパスで命令処理を行えば，MIPS16を採用する場合のハードウェアの追加を最小限に抑えることができる(図38)。

しかし，この方式を採った場合，パイプラインの命令デコードステージが1段増加する。命令の実行ステージより前段のステージを増やすことは，分岐命令の実行クロック数(遅延スロットに相当するアイドル期間)を増やすことを意味する(図39)。当然，命令実行のスループットは低下し，処理性能の低下を招く。MIPS16が命令コードサイズの縮小にあると割り切れば，こういう実装方式も正解である。処理性能を第1に考えれば，命令のデコードを1段階で行うべきである。しかし，この場合はデコード回路が複雑になり，動作周波数を上げにくいという欠点がある。先に挙げたThumbではパイプラインの2ステージを使用して命令デコードを行っていた。そのせいか，LSIロジックはMIPS16の標準的な実装方式についても2段階のデコードを提唱していた(ように記憶している)。

新しい命令セット

(1)MIPS32/MIPS64

MIPSの命令セットアーキテクチャであるMIPS I～Vを，MIPSがこれから注力していこうとしている組み込み制御分野に適合するように見直しをしたのがMIPS32，MIPS64という命令セットである。

MIPS32はMIPS I，MIPS IIのスーパーセットで，従来の命令セットのうち，32ビットモードの命令を基準にしながら，メモリ管理や特権モードはR4000やR5000で用いられている命令を採用して定義されている。メモリ管理を簡単にするために，MMUがない構成や，TLBの代わりにブロックアドレス変換(BAT)と呼ばれる簡略版のMMUを採用することも可能である。通信やマルチメディアのアプリケーションでのデータのスループットを向上させる目的で，MIPS IVから条件転送命令やプリフェッチ命令が採り入れられている。また，AV機器やマルチメディア用途で必要になる固定小数点(早い話が整数だが)専用のDSP機能の命令が新設された。具体的には，汎用レジスタに結果を格納する乗算(MUL)，積和演算(Multiply-Add/Multiply-Subtract：MADD，MADDU，MSUB，MSUBU)と後方1/0

図37　MIPS16のオペコードマップ

メジャーオペコード(op)のビットエンコーディング

命令ビット[15:14]	命令ビット[13:11] 000	001	010	011	100	101	110	111
00	addiusp[注1]	addiupc[注2]	b	jal(x)[注3]	beqz	bnez	SHIFT	*ld*
01	RRI-A	addiu8[注4]	slti	sltiu	I8	li	cmpi	*sd*
10	lb	lh	lwsp	lw	lbu	lhu	lwpc	*lwu*
11	sb	sh	swsp	sw	RRR	RR	extend	*I64*

注1. addiusp：addiu rx, sp, immediate
2. addiupc：addiu rx, pc, immediate
3. jal(x)　：jal命令およびjalx命令
4. addiu8　：addiu rx, immediate

RRマイナオペコード(RR-タイプ命令)

命令ビット[4:3]	命令ビット[2:0] 000	001	010	011	100	101	110	111
00	j(al)r[注5]	*	slt	sltu	sllv	break	srlv	srav
01	*dsrl*[注6]	ϕ	cmp	neg	and	or	xor	not
10	mfhi	*	mflo	*dsra*[注6]	*dsllv*	*	*dsrlv*	*dsrav*
11	mult	multu	div	divu	*dmult*	*dmultu*	*ddiv*	*ddivu*

注5. j(al)r　：jr rx命令(ry=000)
jr ra命令(ry=001, rx=000)
jalr ra, rx命令(ry=010)
6. dsrlとdsraはrxレジスタフィールドを使用して，シフト数(0は8桁のシフト)をエンコードする．この2つの命令の拡張版ではEXT-SHIFT64形式を使用する．この2つのRR命令だけを拡張できる．

RRRマイナオペコード(RRR-タイプ命令)

命令ビット[1:0] 00	01	10	11
daddu	addu	*dsubu*	subu

RRRI-Aマイナオペコード(RRI-タイプADD命令)

命令ビット[4] 0	1
addiu[注7]	*daddiu*[注8]

注7. addiu　：addiu ry, rx, immediate
8. daddiu：daddiu ry, rx, immediate

SHIFTマイナオペコード(SHIFTタイプ命令)

命令ビット[1:0] 00	01	10	11
sll	*dsll*	srl	*sra*

I8マイナオペコード(I8-タイプ命令)

命令ビット[10:8] 000	001	010	011	100	101	110	111
bteqz	btnez	swrasp[注9]	adjsp[注10]	*	mov32r[注11]	*	movr32[注12]

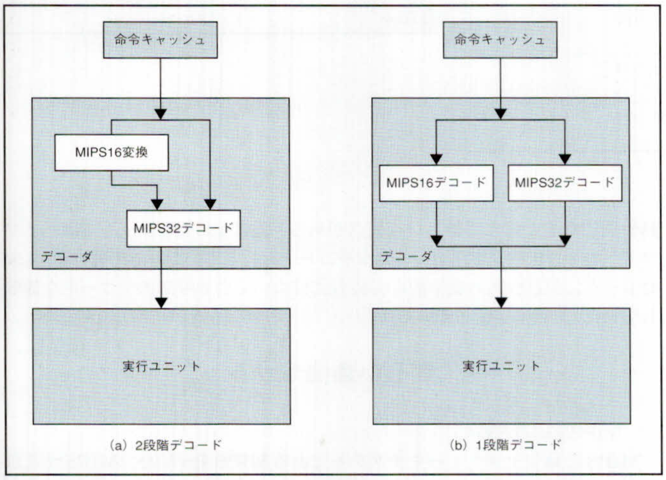
注 9. swrasp：sw ra, immediate(sp)
10. adjsp　：addiu sp, immediate
11. mov32r：move r32, rz
12. movr32：move ry, r32

I64マイナオペコード(64ビットのみ，I64-タイプ命令)

命令ビット[10:8] 000	001	010	011	100	101	110	111
ldsp[注13]	*sdsp*[注14]	*sdrasp*[注15]	*dadjsp*[注16]	*ldpc*[注17]	*daddiu5*[注18]	*dadiupc*[注19]	*dadiusp*[注20]

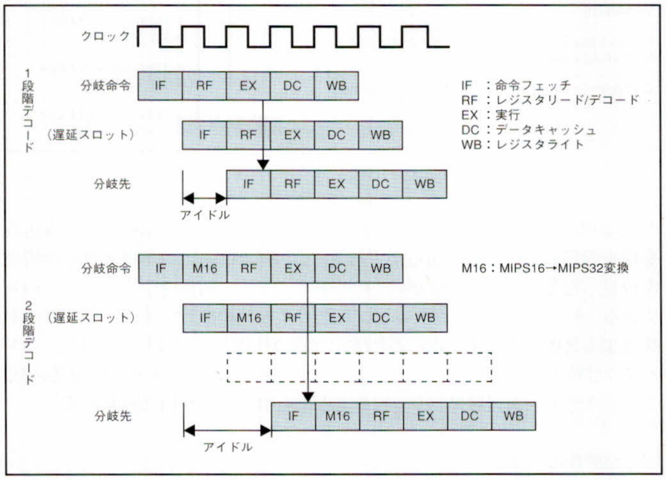
注 13. idsp　：ld ry, immediate
14. sdsp　：sd ry, immediate
15. sdrasp：sd ra, immediate
16. dadjsp：daddiu sp, immediate
17. ldpc　：ld ry, immediate
18. daddiu5：daddiu ry, immediate
19. dadiupc：daddiu ry, pc, immediate
20. dadiusp：daddiu ry, sp, immediate

図38　MIPS16の2段階デコード



(a) 2段階デコード　　(b) 1段階デコード

図39　2段階デコードによって発生する遅延スロット



検出(Count Leading Zero/One：CLZ/CLO) 命令である。

図40に新規命令のオペコードマップを示す。ところで，これらの命令機能はすでに各社が独自に実装していて，同じMIPS系MPUでも互換性がない部分だった。MIPSが正式に命令セットに組み込むことでプログラムの互換性が生まれることが期待される。また，MIPS32では，装置のデバッグ用にEJTAGと呼ばれるICE機能も標準装備している(トレース機能がないという噂だが，使いものになるのか)。

MIPS64はMIPS IV，MIPS Vのスーパーセットである。MIPS32とも下位互換があり，64ビット命令と64ビットの仮想アドレス空間をサポートしている。今後，MIPSアーキテクチャを採用するMPUはMIPS32またはMIPS64のいずれかをサポートすることになるであろう。

(2) MIPS-3D ASE

MIPS-3D ASE(Application Specific Extension)は，3次元グラフィックアプリケーションを高速に処理するためにMIPS64の仕様を拡張したものである。3次元ジオメトリ処理専用に，従来の命令セットに新たな13命令を追加した。MIPS Vのスーパーセットになっている。もともとMIPS Vは固定小数点や浮動小数点をDSP形式で処理するために強化されたものである。MIPS64の浮動小数点命令は，単精度，倍精度，ペアドシングル(Paired Single)のデータ型を処理できる。ペアドシングルとは，1個の64ビットレジスタに2つの単精度浮動小数点データを格納し，2ウェイのSIMD(single instruction multipledata)方式での処理(要は並列実行)を可能にするデータ型である。

図40　新規命令のオペコードマップ（不完全版）

メインオペコードマップ

Opcode[28:26] (columns), Opcode[31:29] (rows)

0								
1								
2								
3					Special2			
4								
5								
6								
7								

Special2 サブマップ

Opcode[2:0] (columns), Opcode[5:3] (rows)

0	MADD	MADDU	MUL		MSUB	MSUBU		
1								
2								
3								
4	CLZ	CLO			DCLZ	DCLO		
5								
6								
7								

図41　MIPS-3Dで拡張された命令

ニーモニック	処理
ADDR	浮動小数点リダクション加算
MULR	浮動小数点リダクション乗算
RECIP1	逆数。リダクションされた精度
RECIP2	逆数。第2ステップ
RSQRT1	平方根の逆数。リダクションされた精度
RSQRT2	平方根の逆数。第2ステップ
CVT.PS.PW	ペアドワードからペアドシングルへの型変換
CVT.PW.PS	ペアドシングルからペアドワードへの型変換
CABS	浮動小数点絶対値比較
BC1ANY2F	多重条件コード分岐
BC1ANY2T	〃
BC1ANY4F	〃
BC1ANY4T	〃

MIPS-3D命令を用いると，もっとも内側の処理ループのコードサイズを30%削減できるので，1秒間に処理できるポリゴン数が45%増加するそうだ。頂点の座標変換での行列の乗算を高速化するために，ペアドシングルリダクション加算命令（ADDR）が定義された。画像のクリッピングは，ペアドシングル絶対値比較命令（CABS）と多重条件コード分岐命令（BC1ANYnx）によって簡略化できる。透視変換には逆数命令（RECIP1，RECIP2）が使用できる。光源処理には逆数平方根命令（RSQRT1，RSQRT2）が使用できる。また，ペアドシングルのほかにペアドワード（Paired Word）というデータ型も定義されていて，ペアドシングルとペアドワード間でデータ変換を高速に行う命令（CVT.PS.PW, CVT.PW.PS）もある。ペアドシングル間の加減乗除，平方根演算などを行う命令は，元々MIPS Vで定義されていた。追加された13命令の概要を図41に示す。

MIPS-3Dの応用分野としては，デジタルセットトップボックス，ゲーム機，PDA，DVDなどの低価格な組み込み分野での3次元グラフィック処理が想定されている。MIPSの性能試算では，500MHz動作のMPUで，秒間2500万ポリゴン（光源つきだと1000万ポリゴン）を処理できるそうである。

この命令セットを最初に実装する最初のMPUは，今年中に発表されるといわれているRuby（20Kc）である。

おわりに

MIPS系MPUでの仮想記憶管理と特権命令，FPUの動作について解説してきた。ついでに，MIPS16とMIPS32/64という新しい命令セットについても簡単に触れた。前回の命令セットの説明と併せて，MIPSアーキテクチャの全貌がつかめたであろうか。基本的なキーワードは押さえたつもりである。最近，MIPSアーキテクチャを持つIPコアがいくつか発表されているが，それらのニュースリリースを読むときの参考にしてほしい。さて，この連載は今回で「第1部完」である。しかし，新たなネタが見つかったら次号から第2部が始まるかもしれない。

追伸

本稿は「初めて読む」を謳っているため，参考に「はじめて読む486」という本を読んでみた。この本の大半のページはインテルのi486のMMUの解説で占められている。資料的価値は十分だが，「はじめて読む」というレベルを遥かに超えていると思うのは私だけか。この連載のレベルは読者にとってどの程度なのか気になるところである。

次々と発表されるMIPS系IPコア。しかし……

最近，MIPSアーキテクチャのIPコアが次々と発表されている。まずはJADE（4Kp，4Km，4Kc）。これは本家本元であるMIPSが発表したCPUコアである。新しい命令セットであるMIPS32を実装する最初のMPUである。取り立てて注目する特徴はない（低消費電力を謳っているがキャッシュの電力が加算されていないのであまり意味がない）が，チップとしての供給ではなく論理合成可能なHDL記述でリリースされるところが新しい。MPUの設計でいちばん苦労するのは論理合成後のレイアウト以降の過程であるので，お気楽なリリース形態であるといえなくもない。1チップとしては製造ライセンスを獲得しているTI（Texas Instruments）社が供給することになっているという。

また，MIPSはマイクロプロセッサフォーラム'99で，JADEの後継であるOPAL（5Kc）の概要を発表した。これはMIPS64を実装する最初のMPUである。400MHzで動作させるために，パイプラインの命令デコードステージを2段にしたとか。しかし，パイプライン図を見ると，データキャッシュへのアクセスが1.5段分しかなく，そこがスピードネックになりそうである。まあ，300MHzで動作すれば上出来というところ。これも，おそらくTIがチップ製造を行うことになるであろうが，頑張ってほしい。

R5400，SR1コアの開発で一躍有名になったSandCraft社がSR1ファミリの第2弾として，SR1-GXというIPコアを発表した。デジタルセットトップボックスやゲーム機用に高速3次元ジオメトリ処理や音声構成のための浮動小数点演算機能を強化している。Vector3Dという技術がそれで，MIPSの命令セットにはない独自の命令セットである。2ウェイスーパースカラアーキテクチャを採用し，400MHz動作で，800 MIPS，1.6GFLOPSの性能という。SR1-GXのアーキテクチャはLSIロジックにライセンスされているので，実際のチップは同社から発表されるのであろう。そのLSIロジックからはRISCore 64600というMPUが発表された。ニュースリリースによると，どうやらSandCraftのSR2（SR1-GXのことか）のカスタム版（バス仕様を従来のIDT社のMPUとコンパチにした？）だそうである。命令キャッシュとデータキャッシュが，それぞれ64Kバイトの容量というのがすごい。個人的にはSandCraftという会社にはいい印象を持っていないが，LSIロジック社にとっては頑張ってほしい（こればっかり）。

MIPSもそうだが，SandCraftなど，論理設計を売りにしている会社は，半導体製造技術に過大な期待を寄せているきらいがある。チップにしたときに，論理設計（予定）どおりの周波数が簡単に実現可能だと思っているとしたら甘い。苦労するのはチップの製造メーカーなのである。

simips簡易マニュアル

中森 章　Nakamori Akira

前回の1999夏号では，MIPS Rシリーズのアセンブラ/逆アセンブラを作成して紹介した。アセンブラと逆アセンブラだけでは実際のプロセッサの動作などを把握しづらいということもあり，今回はMIPSアーキテクチャの動作理解の助けとなるように，ソフトウェアシミュレータを作成してみた。CPUの動作をかなり正確にシミュレートすることができる。これらのシステムだけでもRシリーズのCPUシステムを開発することが可能だろう。

CD-ROMに収録されている実行形式はWindows用であるが，ソースは単純な(?)C言語なので他機種への移植(というか再コンパイル)は容易なはずだ。

MIPS IVレベルまで実装されているので，特殊な命令(5ページ参照)さえ使わなければ，PlayStation2相当のソフトウェア開発も可能になる……かもしれない。

概要

R4100とR4300のアーキテクチャを基に，MIPS IV，MIPS16，MIPS32の命令セットを追加してある。64ビットモードにも対応している。

ロードできるプログラムの形式は，

(1)インテルHEX形式
(2)モトローラHEX(Sレコード)形式
(3)アセンブラソース形式(文法は1999夏号のアセンブラと同じ)
(4)メモリイメージ形式(1999夏号のアセンブラが標準出力に出力する形式)をサポートしている。プログラムをロードしたら，resetコマンドを入力し，

t 1000

などと入力すれば，プロセッサの動作をトレースできる。

コマンド一覧

●q

シミュレータの終了。

●reset

プロセッサにリセットをかける。具体的にはPCが0xbfc00000に設定され，ステータスレジスタのERLビットがセットされるだけ。MIPSの仕様にはないが，キャッシュも無効化する。

●t 回数

PCが示すアドレスから命令をトレースする。「回数」を省略すると1命令のみをトレースする。

●disp ユニット

tコマンド実行時にレジスタの値をダンプするユニット(cpu，cp0，fpu)を指定する。「all」を指定するとすべてのユニットのレジスタをダンプする。「inst」を指定すると逆アセンブルコードを表示する。

●nodisp ユニット

dispコマンドの逆。tコマンド実行時にレジスタをダンプしないユニットを指定する。既定値は，

disp inst
disp cpu
nodisp cp0
nodisp fpu

である。

●cpu 種類

シミュレートするCPUの種類(R4100，R4300)を指定する。「R4100」を指定するとMMUのページサイズに1Kバイトが指定できる。「R4300」でのページサイズは4Kバイト以上である。それに伴い，CP0のEntryLo0/EntryLo1レジスタに設定する値が異なってくる。詳しくは，各プロセッサのユーザーズマニュアルを参照のこと。「種類」を省略すると，現在の設定を表示する。既定値は，

cpu r4300

である。

なお，マニュアルはNECのサイト(http://www.ic.nec.co.jp/micro/index.html)からダウンロードできる。

●a アドレス

インラインアセンブラ。文法は前回(1999夏号)で示したのと同じ。「アドレス」は仮想アドレスを想定しているが，下位29ビットを物理アドレスとみなして，その物理アドレスの位置にアセンブルしたコードを格納する。

●a16 アドレス

インラインアセンブラ。MIPS16用。文法は前回(1999年夏号)で示したのと同じ。

●u 仮想アドレス

逆アセンブラ。アドレスは仮想アドレスなのでマップ領域(アドレス変換を行う領域)を逆アセンブルする場合は，アドレス変換例外が発生しないようにする必要がある。

●u16 仮想アドレス

逆アセンブラ。MIPS16用。

●li ファイル名 オフセット

インテルHEX形式のプログラムのロード。「オフセット」は省略可能(0が指定されたとみなす)。インテルHEXはi8080からi8086に移行する際にベースアドレス(セグメントアドレス)を指定できるように拡張されたが，これでは1Mバイトの領域しか指定できない。また，インテルHEXを出力するコンパイラなどではベースアドレスが省略されることが多い。「オフセット」はその場合のベースアドレスを指定するために使

用する。本シミュレータでは32ビットでベースアドレスが指定できるように，さらに拡張を行っているが，ローカルな仕様なので詳細な説明は省く。一般的なインテルHEXとは上位（下位？）互換性があるので不都合はないはず。

●ls ファイル名 オフセット

モトローラHEX形式（Sレコード）のプログラムのロード。「オフセット」は省略可能（0が指定されたとみなす）。

●lm ファイル名

メモリイメージ形式のプログラムのロード。1999年夏号でのアセンブラが標準出力に出力する形式である。アドレスとデータ（命令）の組が並んだだけの単純な形式である。

●la ファイル名

アセンブラ記述のプログラムのロード。文法は1999年夏号でのアセンブラと同じ。

●la16 ファイル名

アセンブラ記述のプログラムのロード。MIPS16用。

●r ユニット

指定したユニット（cpu，cp0，fpu）のレジスタの現在の値をダンプする。

●bi0 命令コード メッセージ

命令コードによるブレークポイントの設定。「メッセージ」にはブレークポイントにマッチしたときに表示する単語（1語）を指定する。bi0のみを入力するとブレークポイントの許可／禁止が切り替わる。既定値は，

bi0 0x03fffc0d Break_in_Happy

を指定したのと同等である。これは「break 0xffff0」命令である。

●bi1 命令コード メッセージ

命令コードによるブレークポイントの設定。「メッセージ」にはブレークポイントにマッチしたときに表示する単語（1語）を指定する。bi1のみを入力するとブレークポイントの許可／禁止が切り替わる。既定値は，

bi1 0x03ffffcd Break_in_Sad

を指定したのと同等である。これは「break 0xfffff」命令である。

●ba 仮想アドレス メッセージ

仮想アドレス（下位32ビット）によるブレークポイントの設定。「メッセージ」にはブレークポイントにマッチしたときに表示する単語（1語）を指定する。baのみを入力するとブレークポイントの許可/禁止が切り替わる。既定値はブレーク禁止状態である。仮想アドレスはビット31で符号拡張した値で比較するので64ビットモードで実行中では指定できないアドレスがある（ゴメン）。

●m 物理アドレス データ

「物理アドレス」で指定されるアドレスに1ワード（32ビット）の「データ」を書き込む。

●dw 物理アドレス1 物理アドレス2

「物理アドレス1」から「物理アドレス2」までの物理アドレスで指定されるメモリの内容をワード（32ビット）単位に表示する。

●dh 物理アドレス1 物理アドレス2

「物理アドレス1」から「物理アドレス2」までの物理アドレスで指定されるメモリの内容をハーフワード（16ビット）単位に表示する。

●db 物理アドレス1 物理アドレス2

「物理アドレス1」から「物理アドレス2」までの物理アドレスで指定されるメモリの内容をバイト（8ビット）単位に表示する。

●h

ヘルプメッセージを表示する。

サンプルプログラム

サンプルプログラムとしてDhrystoneベンチマーク（簡略版）を添付した。モトローラHEX形式で格納してある。

Dhry21.sre	**R5000用コード**
Dhry16.sre	**MIPS16用コード**

どちらも，BOOT部分を付加してあるので「ls」コマンドでロードして，「reset」コマンド入力後，すぐに実行できる。このプログラムではキャッシュの初期化部分は省いてある。面倒臭いので「reset」コマンドでキャッシュの初期化をするようにしている。

制限事項

(1) リトルエンディアンのみの対応。リバースエンディアン（ユーザーモードでのエンディアン反転）にも未対応
(2) キャッシュは命令/データともに16Kバイト固定。ラインサイズは4ワード固定。構成はダイレクトマップ，ライトバック方式
(3) キャッシュエラーはサポートしない。キャッシュのパリティビットも省略してある
(4) FPUの演算例外の検出（特にCOP1X）がいい加減。正規化数のみを入力する場合は正常に動作するはず
(5) MMUの動作（というかアドレスエラーの検出）は十分な検証を行っていないので，誤動作の可能性がある。たいていの場合は正常動作するはずだが
(6) 検証は十分に行ったが，まだバグが潜んでいる可能性がある。変な動作をしても目くじらを立てないように。検証はR4100モードで行ったのでR4300モードはちょっと怪しい
(7) R4300モードでバグが発見された。R4300のTLBエントリ数は32個だが，プログラム中ではR4400と同じ48個として実装されている。詳しくはソースリストを参照のこと

改版履歴

Rev.0.90	一応の完成バージョン
Rev.0.91	逆アセンブルを仮想アドレスで行うように改造
Rev.0.92	キャッシュ動作のバグを修正。MMUのマイナーなバグを修正
Rev.0.93	XX=0，KSU=2，XRL=1でMIPS IV命令が例外になるバグを修p。64ビット命令の禁止時，DCLZ，DCLO命令が実行できるというバグを修正

Hardware Geeks

PIC16F84シミュレータ PSIM6

高尾 克彦 TAKAO Katsuhiko

組み込み型小型CPUとして多彩な活用がされているPICマイコン。前回発表したアセンブラ/シミュレータシステムを改良し，より本格的な開発支援環境を作ってみました。ハードウェア開発のお役に立ててください。

1999夏号では，アセンブラのついでにとり急ぎ作ってみましたという感じでCD-ROMに収録したPIC16F84用シミュレータですが，実は原稿を締め切ってから，CD-ROMのマスターアップまで少し時間があり，本文に述べられている状態よりも，もう少し完成度の高いものが収録されています。たとえば，本文にはレジスタ，メモリの内容の変更ができないようなことが書いてありましたが，実際には，それぞれレジスタウィンドウ，メモリウィンドウをクリックすることにより行えます。

さらにもう少しの手直しで，それなりに実用に耐えるのではないかという意見を何人かの方からいただきました。実際，私もそうだと思いますので，少し手を加えて，詳しく説明します(そういえば，三沢氏は，今号でPICマイコンを使った制御ものの原稿を書くとおっしゃってましたが，間に合ってるかな？)。

■プログラム作成の流れ

PICマイコンのプログラムは，以下のような流れで作成します。

1) エディタでソースファイルを作成
2) 1)のファイルをアセンブラPAでアセンブル。文法エラーなどがあれば，1)へ戻る。エラーがなければ，*.LSTファイルと*.HEXファイルが作成される
3) 2)の*.LSTファイルをシミュレータPSIM6で，作成したプログラムにアルゴリズム的な問題がないか検討する。問題があれば，1)へ戻る
4) 2)の*.HEXファイルをプログラムライタを用いて実機に焼き込む

純正のツールを用いると，プロジェクトだとかリンカなどが絡んできて，もう少し複雑なのですが，PICマイコンにそこまで本格的なものよりも，デバイス同様手軽に使えることを目指して作成した開発ツールですので，上記の過程で十分です。

■ソースファイルを作成

PICマイコンを作っているMicrochip Technologyでは，データシート中，アセンブラニーモニックも規定しているのですが，初心者にはわかりづらく(実は，私はもはや初心者ではないはずですが，1カ月も離れると，細かいところはかなりあやふやになります)，不親切なので，1999夏号で発表した私家版アセンブラのニーモニックを使用して，以下，説明します。

とりあえず，リスト1をcount.asmというファイルネームで作成してください。これは，RAM上の0x0c番地の値を0にセットし，その後，ひとつ増やして1にするというプログラムです。

1行目：この行以下の設定は3つあるメモリ空間(ram, program, eeprom)のうち，ramに対するものであることを宣言
3行目：以下の命令・定数は0x0c番地から行うという疑似命令(PIC16F84では，0x00～0x0bまでは，システム予約なので，一般用途には使用できない)
5行目：このアドレス(ram空間，0x0c番地)に，"count"というラベルを設定
7行目：この行以下の命令は，programメモリ空間に対するものであることを宣言
9行目：(PICマイコンの命令)ラベルcountの内容を0にする
10行目：(PICマイコンの命令)ラベルcountの内容をインクリメント(1増やす)する
12行目：(PICマイコンの命令)ラベルLoopに飛ぶ(つまり，12行目を永遠に実行する)

という感じです。ちなみに，PICマイコンでは，HALTに相当する命令がないので，PSIM6では，リスト1の12行目のように同じアドレスで永久ループに入る場合，"Run"の実行を止めるということにしています。

リスト1　count.asm

```
 1: segment ram
 2:
 3: org     0x0c
 4:;
 5:count:  ds       0
 6:;
 7: segment program
 8:;
 9: clrcount
10: count++
11: Loop:
12: goto   Loop
```

リスト2　count.1st

```
      :          segment ram
      :
      :          org     0x0c
      :     ;
R000c :     count:       ds      0
      :     ;
      :          segment program
      :     ;
 0000:0190       clr     count
 0001:0a90       count++
      :     Loop:
 0002:2802       goto    Loop
```

■アセンブル

では，リスト1のファイルを実際にアセンブルしてみましょう。

```
D:>pa count.asm
0 Error(s), 0 Warning(s)
D:>
```

という具合に，エラー，警告ともに表示されていませんね？　どちらかが表示された場合，入力間違いですので，よくリスト1と見比べてください。

アセンブルが成功すると，リスト2のような*.LSTファイルが作成されるはずです。これは，リスト1の結果を，PICマイコン用にアセンブルした結果です。基本的に，

アドレス：データ＋ソースファイルの記述

という具合になっています[*1]。

*1：ほかのシステムに慣れ親しんだ方は，PAが，OBJファイルも，SYMファイルも作成しないのに，PSIM6にデバッグ情報を渡していることを不思議に思うかもしれませんが，実は，PAの出力したLSTファイルを，PSIM6はそのまま読み込んでいます。

■シミュレータ

いよいよシミュレータを使用します。Windowsから，PSIM6を起動し，メニューからFile｜Openを選び，先ほどのcount.lstを選択します（画面1）。このままでもプログラムがなにを行っているかを知ることはできるのですが，ほかの情報も表示させましょう。メニューから以下のアイテムを選択してください。

Window｜Register
Window｜Ram

それぞれ，PICマイコンのレジスタの値，RAMの値が表示されているはずです。画面中，赤く表示されている項目は，直前の動作で変更されたことを表しています。この場合ですと，ファイルを読み込んだ時点で，（PICマイコンの）電源リセットにより初期化されたレジスタやメモリエリアです。また，ソースプログラムウィンドウの暗い黄緑色の線は，次に実行するべき命令を表しています（以下，〝現在行〟）。いまは，電源リセット直後ですから，プログラムの先頭0x0000番地を指しています。

これで，下準備は完了です。これから，以下の3種類の方法で，デバイスの動作をシミュレーションすることができます。

●Debug｜Run（あるいはF5キー）
現在行から，プログラムを実行します。
・メニューからDebug｜Stopを選択
・同じアドレスで永久ループを行う命令を実行
・PCがブレークポイントに到達
のうち，いずれかで，実行を中断します

●Debug｜Trace（あるいはF11キー）
現在行から，1命令だけ，実行します。Call命令をTraceした場合は，サブルーチンの先頭に現在行を設定します

●Debug｜Step（あるいはF10キー）
Traceとほぼ同じですが，サブルーチン中は，トレースしません。つまり，call文をStepした場合，プログラムは，サブルーチンからリターンするまで，PSIM6は，プログラムを実行します

たとえば，

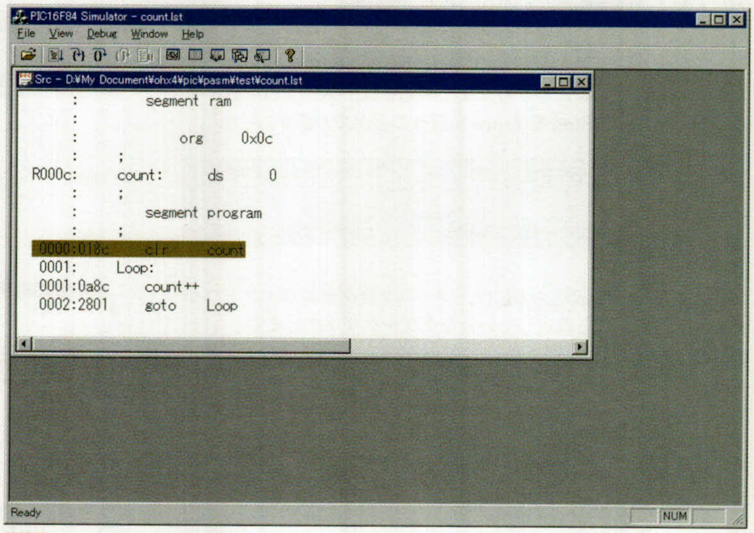
```
1: call SetWZero
2: w = w + 01
3: goto .    // 永久ループ
4: SetWZero:
5: w = 0
6: return
```

というプログラムで，現在行が1にあった場合，

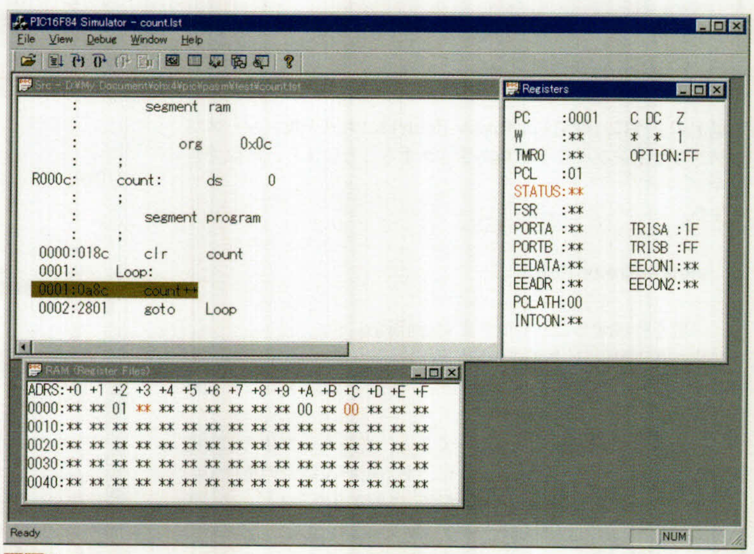
●Traceを行うと，1行目をシミュレートし，現在行を5行目に移動
●Stepを行うと，1，5，6行目をシミュレートし，現在行を2行目に移動します。

では，実際に使ってみましょう。先ほど，アセンブラで作成したCount.lstは，シミュレータに読み込まれていますか？

キーボードのF10（Step）を押してください。ソースプログラムウィンド



画面1　count.lstを読み込んだところ



画面2　とりあえず，RAMウィンドウとレジスタウィンドウを開いてみる

ウの現在行がひとつ下へ移動したことを確認してください。また，レジスタウィンドウのいくつかの項目の値が変わり，表示が赤くなっていますね（画面2）。この赤くなった値は，更新が行われたことを示しています。

もう一度，F10を押して，〝count++〞を実行します。この命令は，RAMの0x0c番地（=count）の値をインクリメントしますので，RAMウィンドウの0x0c番地の値が赤くなって，0x00から0x01になるはずです。

また，レジスタウィンドウやRAMウィンドウの値は，マウスでクリックし，キーボードから値を入力することも可能です。数値を入力しリターンキーを押すか，ほかの項目にマウスを移動すると値が確定します*2。

*2：前回も説明しましたが，PICマイコンは，PCも含めてほとんどのレジスタがRAMにマッピングされています。慣れないうちは，RAMウィンドウの0x00～0x0bは，自分で変更しないほうがよいでしょう。

■Break Point

以上のようにして，Stepを繰り返せば，プログラムの流れはひととおり確認できるのですが，プログラムが大きくなってくると，すでに実証済みの部分が多くなり，その部分をStepするのが面倒になってきます。また，Runを使うと，プログラムを一気に実行してしまうので，途中でなにが起こったのかよくわかりません。

そのようなときに使用するのがBreak Pointです。プログラムのシミュレーションを行いたい場所にBreak Pointをセットし，Runすれば，そこにたどり着くまでの過程は一気に実行され，そこからじっくりとStepやTraceを行うことができます。

	十分に検証した部分	↑Runでいっきに実行↓
Break Point→		
	これから検証する部分	↑Stepでじっくり検証↓

Break Pointを設定するには，ソースプログラムウィンドウから，Break Pointを設定したい行をダブルクリックします。

先ほどのCount.lstがまだ，シミュレータに読み込まれている状態で，

1） Debug｜Reset
2） ソースウィンドウ中，〝goto Loop〞をダブルクリック
3） Debug｜Run

すると，プログラムは〝goto Loop〞まで実行されているはずです。

■Break Pointの回数

以上のように，設定したBreak Pointは，もう一度，ソースプログラムウィンドウをダブルクリックすることにより，解除できます。

また，

View | Break Point

で，一覧を作成することもできます（画面3）。

設定できるBreak Pointの数に制限はありません。極端な話，プログラムのすべての行に，Break Pointを設定することも可能です。

さて，プログラムが込み入ってくると，あるループ中の120回目に問題が発生する，というように，ある場所&特定の条件でのみ発生するバグが出る場合があります。このようなときに，Break Pointを用いると，119回F5を押して，120回目には，忘れずにレジスタ情報，メモリ情報などに注目し，1行ずつStepを行うようにしなければなりません。ちなみに，120回目に間違って，Runをしてしまったら，泣く泣くリセットし，再び119回F5を押さねばなりません。

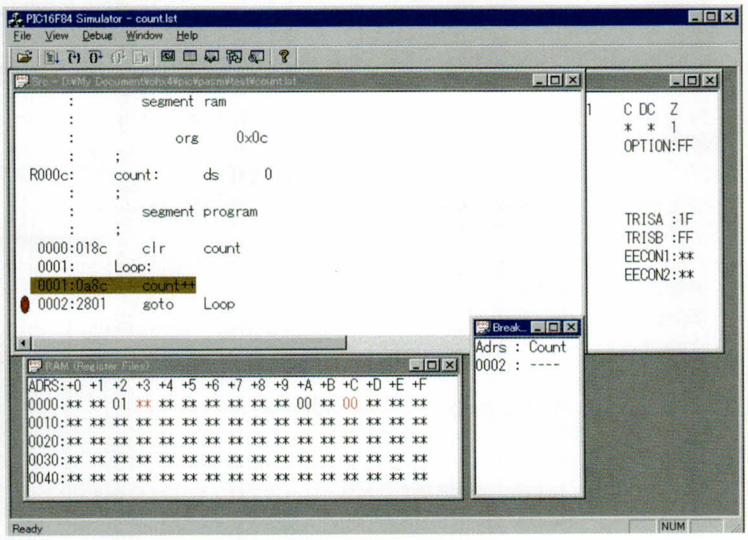
そのようなときは，Break Pointに回数の指定をします。

リスト3は，RAMの0x0c番地に0を代入し，そのあと，0x0c番地の値をインクリメントし続けるというプログラムです。リスト1のときと同様に，アセンブルし，シミュレータに読み込み，

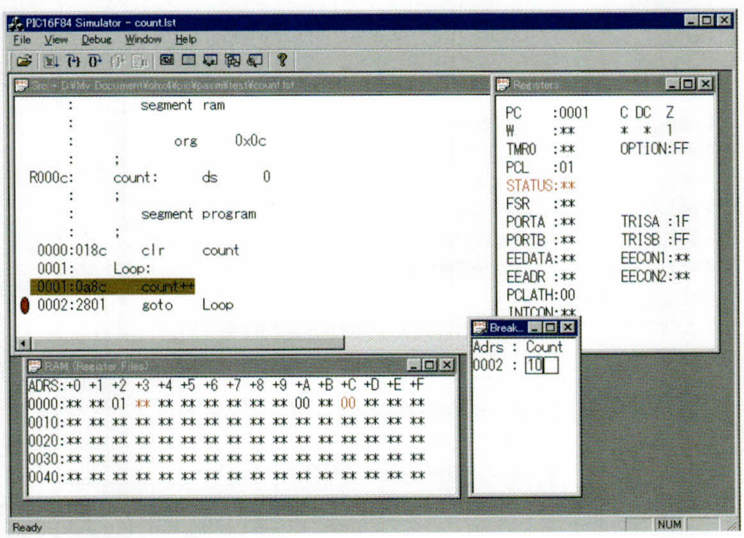
1） Debug｜Reset
2） 12行目の，〝goto Loop〞をダブルクリックし，ブレークポイントを設定
3） View｜Break Point　Break Pointウィンドウを表示
4） Break PointウィンドウのCountの項を，マウスでクリックし，10を代入（画面4）
5） Debug｜Run　プログラムを実行
6） View｜RAM　0x0c番地の値を確認（→10ですね）

ここで，RAMの0x0c番地の値が10だったということは，0，1，2，3……10と10回インクリメントされた証拠です。つまり，PCがBreak Pointに10回ヒットしてから，Runの実行が中断されたということです。

試しに，1）～4）をもう一度行い，今度はRunでなく，Stepを繰り返し，



画面3　ソースウィンドウ中で，ダブルクリックし，Break Pointを設定



画面4　Break Pointにカウントを設定

Break Pointに到達するごとに，Break Pointウィンドウ中のCountが1ずつ減ることを確認してください。

■コンソールウィンドウ

以上の機能で，必要なときに必要な情報（レジスタやメモリの値）を知ることができるようになったのですが，メモリやレジスタの値そのものよりも，それらの和や差が知りたいときが，プログラム作成中にはよくあります。

そんなときには，コンソールウィンドウを使用します。

Window | Console

まず，まっさらなウィンドウが表示されますので，マウスを左クリックするか，キーボードのEnterキーを押します。テキストフィールドが開かれるので，ここで，

w + 2

と入力してください。

リスト3　loop.asm

```
    segment ram

    org     0x0c
;
count: ds      0
;
    segment program
;
    clr count
Loop:
    count++
    goto    Loop
```

これで，画面5のようにwレジスタの値に2を加えたものが，右側に表示されるはずです。

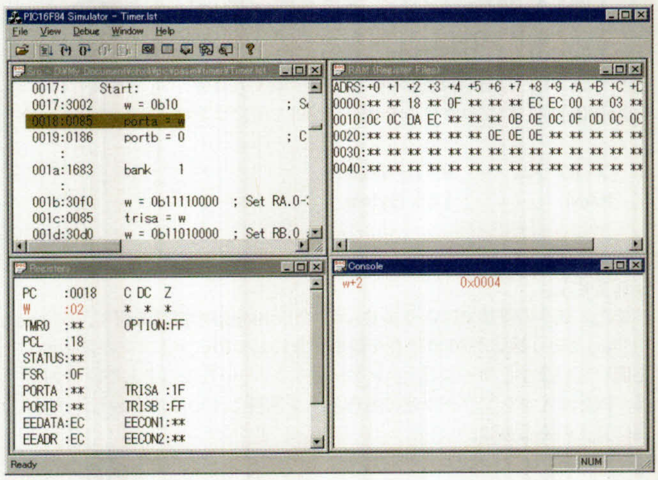
もちろん，

(w + 2) * 4

のような表記も可能です。

RAMの値を表示させたいときには，ラベルの頭に「*」をつけてください。

例）* count



画面5　コンソールウィンドウに「w + 2」を入れてみる。レジスタウィンドウ中の「w」より，確かに2だけ大きい値が表示される

Microchip Technologyから新しいデバイスが出ました　Column

少し前の話になりますが，8月（だったかな？）にMicrochip Technologyから，PICマイコンの最上位ファミリPIC18xxxxファミリが発表されました。

同社のWebページhttp://www.microchip.comによると，コア自身のスペックは，

プログラム空間	2MB
RAM空間	4KB
命令数	75
クロックスピード	40MHz（10MIPS）

だそうで，かなりのことができそうです。組み込みマイコンですから，実際には表1のような周辺回路をまとめたものが，第1弾として，アナウンスされました。

プログラム空間8Kwordというのは，単純計算で，アセンブラで約8000行書けるということで，プログラマには大変そうです[*3]。しかし，PIC18xxxxで拡張された命令はC言語によくなじむそうで，システム開発は，アセンブラでなくC言語を用いても十分実用的である，と同社のホームページに書いてありました。

どうやら，いままでのPICマイコンのターゲット「マイコンを使うのは大げさだし，PLDでは手に余る」とは，狙っているところが異なるようです。

ここまで高性能になってくると，ほかの組み込み用マイコン（日立のH8シリーズなど）などとも競合するので，今後，同社がどのようにラインナップを展開するのか興味深いところです。私のようなホビイストとしては，今後とも，大口・小口・アマチュアに分け隔てなく販売し続けてくれることを願っています。特に，マスクROMを起こせないアマチュアとしては，フラッシュメモリ内蔵ファミリがありがたいところです。

いま時分，40ピンDIPのパッケージを生産してくれているというのも，ハンダづけに高価な機材を使用できないアマチュアにとっては，嬉しいですね。

*3：Z80で8KBというと，約2000～3000行のアセンブラプログラムでしたが，これは，1ニーモニック＝1～4バイトというZ80の命令体系によるものです。PICマイコンの場合，ほぼ1ニーモニック＝1バイトなので，8KBは，アセンブラで約8000行となります。

表1　PIC18xxxxファミリのラインアップ

デバイス	プログラムメモリWord	RAMバイト数	スピードMHz	PIO	ADC10-bit	SIO	PWM	Compactor	Timer	パッケージ
PIC18C242	8192	512	40	23	10	USARTSPII2C	2	-	4 + WDT	28DIP28SOIC
PIC18C252	16384	1536	40	23	10	USARTSPII2C	2	-	4 + WDT	28DIP28SOIC
PIC18C442	8192	512	40	23	10	USARTSPII2C	2	-	4 + WDT	44TQFP40DIP44PLCC
PIC18C452	16384	1536	40	23	10	USARTSPII2C	2	-	4 + WDT	40DIP44PLCC44TQFP

このときのアドレス指定は，ラベルでなくてもかまわないので，

例）＊（count + 2）

のような表記も可能です。

コンソールウィンドウからある項目を除去するには，テキストフィールドの内容を空文字列にしてください（画面6-a，6-b，6-c）。

■PIC@Internet?

Microchip Technology社のホームページには，PICマイコンを使ううえでの有益な情報がいろいろアプリケーションノートとして公開されているのですが，つい前日，アプリケーションノート#724 "Using PIC Micro R MCUs to connect to Internet via PPP" が公開されていました。さすがにFTPやHTTPやTCPなしの本当にPPPだけなのですが，自分でダイアルアップし，LEDを制御できるマッチ箱大の装置が紹介されています。C言語で作成したプログラムをPIC16C63A用にコンパイルして，

プログラム	2170 Word
RAM	145 Bytes

だそうですので，PIC18xxxxを用いれば，さらに凝ったことができるかもしれません。

また，日本の雑誌ではありませんが，Microchipの協賛を得て，「Circuit Cellar」という雑誌が2000年春を締め切りにしてPICマイコンのコンテストを開いています。テーマは「インターネット」……ひょっとして，今回こそは，PICマイコンでブラウザでも作る人を期待しているのかもしれません。腕に覚えのある方は，

http://www.circuitcellar.com/pic2000

を参照してください。

■次回の予告

現在のバージョンでは，メモリやレジスタの値指定に，即値しか使えません。少し不便ですので，なんとかサポートしたいと思っています。

Break Pointの設定も回数だけでなく，条件つきで実行をやめるとか，そのまま続けるかを設定できるとよいなぁ，とも考えています。たとえば，0x30番地の値が15以上になったときにブレイクするとか。

また，ぜんぜん機能とは関係ありませんが，ようやくスプラッシュウィンドウの出し方をマスターしたので，これもそのうち，実装したいと思っています。

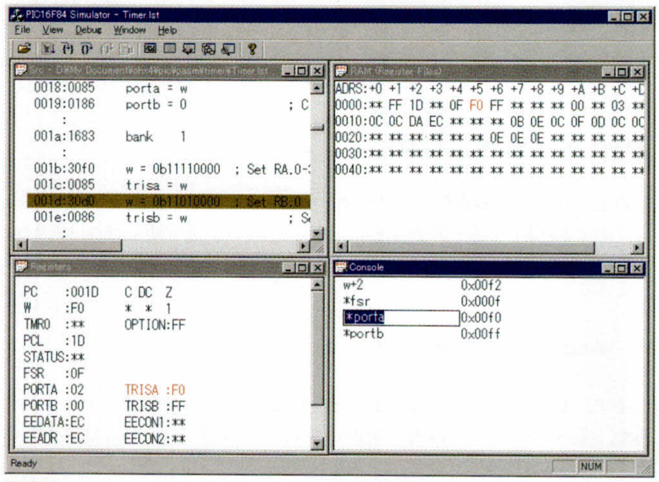
等々，構想は膨らむのですが，やはり時間との戦いですね。詳しくは次回（最終回の予定）。

PASM Column

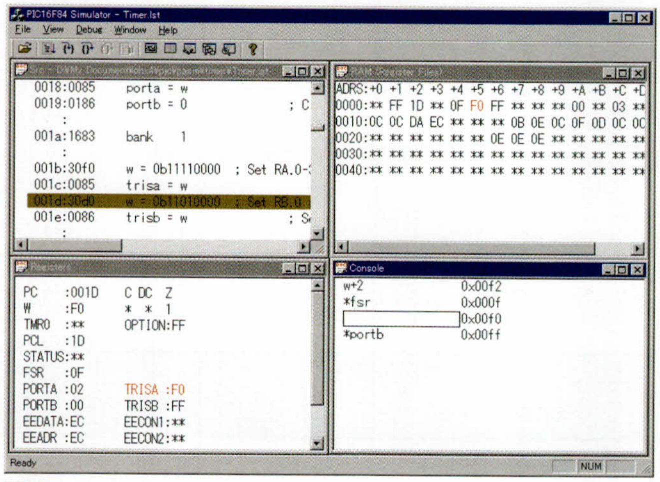
今回，PSIMがラベル情報を扱うようになった関係上，PASMも少しバージョンアップしました。

具体的な変更点は，LSTファイルのフォーマットで，ラベルを定義している行では，必ず行の先頭にアドレス情報が表示されるようになっています。

よって，ラベルウィンドウやコンソールウィンドウを使用する場合は，必ず今月号に付属のPASMを使用してください。



画面6-a　コンソールウィンドウ中の"porta"を消すには



画面6-b　テキストフィールドを空文字列にして

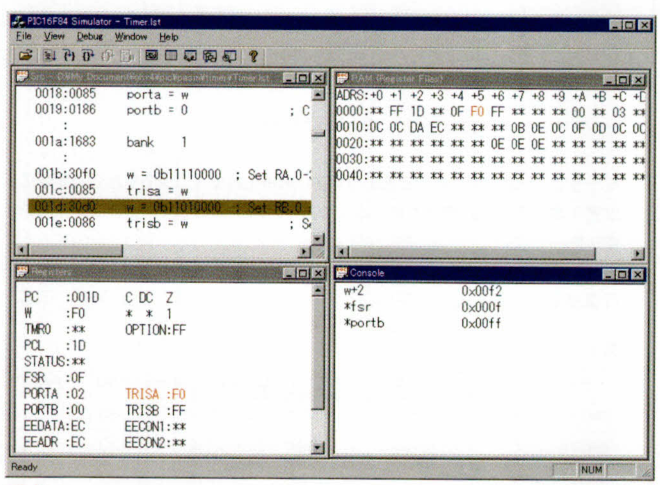


写真6-c　Enterキーを押します

なぜ，「PSIM6？」

Column

PICマイコンのシミュレータだから，PSIMなのはよいとして，6ってなによ，と思われた方もいるかもしれませんが，たいした意味はありません。今回が初めての本格的なWindowsプログラムだったので，PSIM1は，Viewの基底クラスをCEditViewにして……色がつかないじゃないの，ならばPSIM2（基底クラス＝CRichEditView）なら……なんか不安定だぞ，という具合にいまの形になるまでに，プログラムの雛型が6種類あった，というだけのことです（ちなみに，現在は，基底クラス＝CScrollViewで，なんとか落ち着いています）。

試しにMFCの練習プログラムを作ってみましたぁ，という事情がそのまま出ていますが，野沢さんにアイコンを作っていただいて以来，なんとなくまともなアプリケーションに見えるようになってきたので，そのうち名前もかっこいいものを考えます（ヘルプも作らねばいかんなぁ……）。

その他のPICマイコンに関する動き

Column

先月はPICに関する単行本が2冊も発売されました。

●後閑　哲也「PIC活用ガイドブック」
技術評論社　2380円　ISBN4-7741-0921-5 C3055

第1章「PICの概要」～第6章「PICのプログラム開発手順」は，本誌前号の記事とほぼ同じ内容ですが，細部をもう少していねいに説明しています。前号の記事は，2・3種類のマイコンの経験がある人を想定して書きましたが，初心者には少し難易度が高かったかもしれません。そういう方は，本書が，入門にちょうどよいでしょう。特にMicrochip社純正開発ツールのMPLABの使い方など，図をふんだんに用い，ていねいに説明しています。

第7章では，本書オリジナルの電子工作が紹介されています。温度計（PIC16C711），時計（PIC16F84），データロガー（PIC16C73），汎用モーター制御ユニット（PIC16C73），赤外線リモコン送受信機（PIC16F84），ライントレースロボット（PIC16C73）。また，付属のCD-ROMには，これらのソースコード，HEXファイルのほか，プリント基板のパターン図が収録されています。

Microchip社のホームページには，大量にアプリケーションノートが掲載されていて，PICマイコン上級者の方なら，これだけで参考書はいらないはずなのですが，米国以外では入手困難な部品が多く，不便なことが多々あります（2N3906ってトランジスタ知ってる？）。本書の工作例は，すべて著者が秋葉原で部品を買い求め，実際に製作されたもののようです。ですから，PICマイコン上級者でも，ちょっとした外付け回路集として本書は参考になると思います。

●後閑　哲也「楽しく出来るPIC電子工作」
東京電機大学出版局　2700円＋税　ISBN4-501-32050-8 C3055

上記と同じ著者によるPICマイコンの解説書です。ただし，前書がそれなりに（PICは初めてでも，電子工作一般に）経験のあるホビイスト向けに書かれているのに対し，こちらは，中・高校生くらいを読者として想定してるようです。付録には感光プリント基板の作り方が紹介されていたり，そのパターン図を印刷したトレースペーパーが，挟み込まれていたりと，往年の「初歩のラジオ」や「ラジオの製作」を思い出させる作りです。理論よりも実践を重視した構成になっており，PICマイコン自体の説明や，抵抗とはなにか，コンデンサとはなにか，を軽くひととおり説明したあと，すぐに以下の工作例に入ります（使用デバイスはすべてPIC16F84）。電子ルーレット，光線銃による早撃ちゲーム，超音波距離計，リモコンタンク，簡易周波数カウンタ。

使用したプログラムは，付録のミニCD-ROMに収録されています。

ただ著者の方のせいではないですが，せっかくていねいに命令を1つひとつ説明しているのに（純正）ニーモニックがわかりづらく，どうしてもPICマイコンのソフトウェアが難しく思えてしまうのが残念です。先月号の私家版アセンブラを使用していただければ，変な略語と格闘しなくとも，1/3の説明で済むのに……というわけで，石上氏に頼んで，石上氏のホームページに私家版アセンブラ＆シミュレータをアップロードさせてもらいましたので，これからPIC本を書こうとする方は，どうぞご自由にご利用ください。

http://www.02.246.ne.jp/~ishigami/

● Yoshiaki Uchikawa「PICマイコンの開発環境をつくってみよう」
「パーソナルUNIX Magazine」No.3，毎日コミュニケーションズ

PICマイコン自体の説明はともかく，PICマイコンの開発環境をFree BSD上に作ってしまおうという連載です。「この記事はむしろ環境を作るというプロセスを楽しむ立場で書きます。『なぜ，そのようなことをするのか』という点は抜きにして気楽に読んでください」だそうで，おぉ，同じことを考える人がいるものだと思っていたら，なんとOh!X1999年春号の私の記事が紹介されていました。「日本語の資料でこれだけまとまっているものは他には例を見ないでしょう」えへへ。

私の私家版アセンブラも転載されていて，Free BSD上のMS-DOSシミュレータから実行する方法と，UNIX用にリコンパイルされたバイナリがCD-ROMに収録されています。

作者のUchikawaさんへ。アセンブラの転載をした以上，シミュレータが完成したら（Oh!X2000初夏号掲載予定），こちらの移植もお願いしますね。

http://www.picfun.co.jp/

「PIC活用ガイド」「楽しくできるPIC電子工作」の作者の後閑氏のホームページです。このまま書籍にしてもよいのではないかというていねいな説明が掲載されています。

http://www02.so-net.ne.jp/~morioka/cqpic.htm

私は買いそびれてしまったのですが，「トランジスタ技術」誌1999年12月号に掲載された内容らしいです。これは，すごいです。なんとFPGAで，PIC16F84相当を実現されています。作者の森岡氏の技術力にはひたすら脱帽してしまいます。記述はVHDLで，フルインプリメント（と若干の改良）と動作理解用のサブセットの2種類が掲載されています。

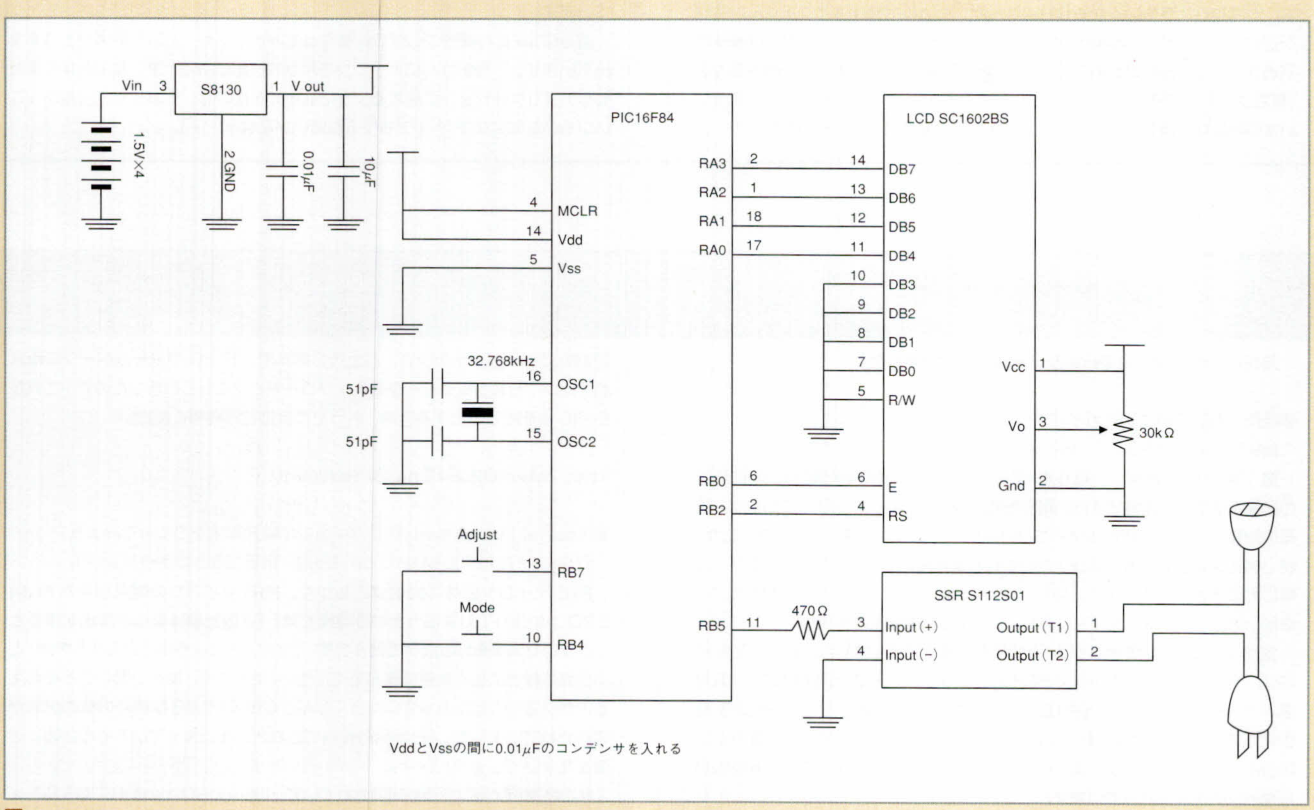
フルインプリメント版をAltera社Flex10Kへインプリメントした場合，700-900LC，サブセット版はその約半分だそうですので，PICマイコンで適当なファミリがなかった場合，このVHDLソースを用いれば，最適なデバイスができそうです（例：PICマイコン core ＋ PIO 8bit x 6 portなど。ただし，EEPROMやADCなどは原理上FPGAに実装できない）。

インターネット上でも，PICマイコンはそれなりに流行ってきているようですし，かなり技術力のある方も多数いらっしゃるようですので，USBインタフェイスつきPICマイコンなんてものが，Microchip社より先に出現するかもしれませんね。

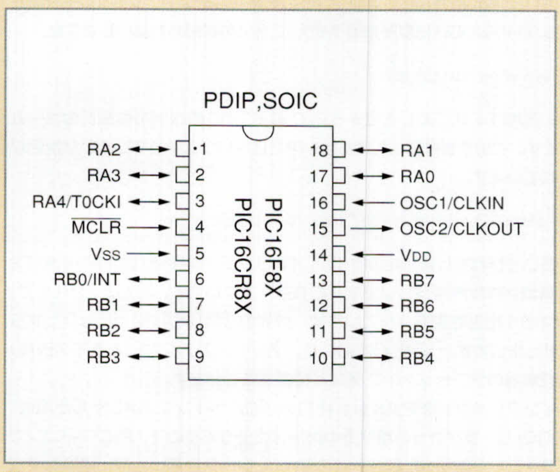
1999夏号の回路図

Column

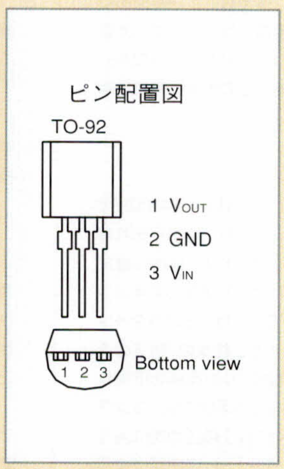
Oh!X1999夏号で製作したプログラマブルタイマですが，肝心の回路図が抜けていました。実際の回路図は以下のとおりです。



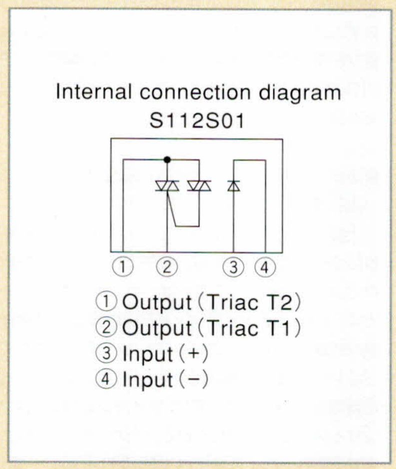
図a　回路図



図b　PIC16F8X端子図



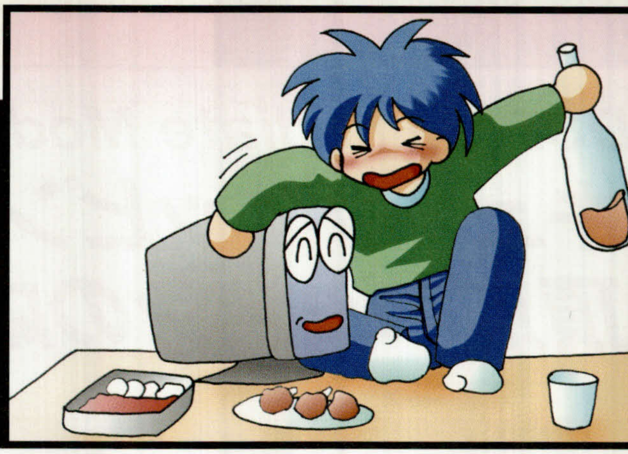
図c　S-813端子図



図d　S112S01端子図

投稿募集

We want you!



明日は自分の手で作れ

ご覧のとおり，Oh!Xは復刊しましたとはいえ，まだまだ至らない点も多く，ジャンルにもかなり偏りが生じています。もっともっと多彩な記事を掲載していくのが「よりよいOh!X」を模索していくうえでも必要と考えています。現状のスタッフもそれなりに強力な面々が並んでいると自負していますが，なにぶん兼業スタッフが多く，どうしても十分な対応が望めません。新しいOh!X構成していくにはまだまだ人手が足りません。新しい戦力を広く募集しています。

●Oh!Xスタッフ募集

前回のスタッフ募集に応募していただいた方すみません。連絡が行き届かずご迷惑をおかけしております。かなり多くの方からご協力申し出のお手紙をいただいたのですが、残念ながらこちらの手不足で管理しきれておりません。

現状では，連絡網の確立やファイルの受け渡しなどを考えるとインターネットアクセス環境必須，またはソフトバンクパブリッシング本社（東京都中央区日本橋箱崎町）まで直接来社可能なことというのが絶対条件になりそうです。なにぶん変則的な体制で編集作業を進めておりますので，ご理解ください。

●それでもスタッフ募集

引き続き各分野で人材を募集しています。あらゆる分野でパソコンおよびさまざまなモノに対してあくなき探求心を持った方を募集しております。とりあえず，「なにかをやりたい」というのが最大のポイントで，さらにいえばより具体的に「～ができる」とか「～がやりたい」と具体的に書いていただけるとこちらも把握しやすいです。ただ，上記のように，インターネット接続環境はほぼ必須と考えてください。

募集するのは基本的には執筆スタッフで，なんらかのテーマに従って記事を書いていただきます。原稿なんか書いたことがないという人でも，経験上，たいていは大丈夫です。プログラミング，ハードウェア制作，ゲームレビュー，ソフトウェアレビュー，その他技術解説などなど，その他，どんな記事でもかまいません。

記事を書くのは自信がないという方ではプログラム制作やビジュアル制作（イラストなど）に絞った仕事をしていただくこともあります。どのような系統のものをやりたいのかなどを明記してご応募ください。

なお，ジャンルについては，現状の記事内容はあまり参考になりません。新しい分野をどんどん開拓していくつもりでいてください。皆様のチャレンジングな企画をお待ちしています。

●投稿募集

スタッフとして参加するのは無理という方でも，なんらかのかたちで協力していただける方を広く募集しています。記事投稿，プログラム投稿なども随時受け付けています。ゲームレビューや単独記事でもかまいません。

プログラム投稿の場合は，機種などはといませんが，対応機種（開発機種），OSのバージョン，ライブラリなどの開発環境のバージョンと実行環境の詳細なども明記するようにお願いします。

ゲームレビュー記事なども募集しています。2号分のレビューを見ていただいた方はなんとなく察しているかもしれませんが，ゲームレビュー記事ではゲームの新旧は問いません（画面写真が撮れる/入手できるくらいのものであることが望ましいが）。もちろん，機種も問いませんし，以前に誌面上で扱われているタイトルでもかまいません。それなりに遊び込んだ作品を，通り一遍でない評価で記事にしてください。複数のタイトルで記事を構成することも可能です。

この辺はちょっと人手不足気味ですので，皆さんの積極的な参加をお待ちしております。

●Oh!X LIVE in '99

音楽データの場合は，多少注意点が異なりますので下記の条項をよく確認してください。曲データのジャンルや種別は特に問いません。ただし著作権の関係で掲載できないデータというものも存在します。これは個別に調査しないと判定できないものが多いので，あらかじめご了承ください。

それでは投稿時の注意です。

まず，使用する機材の構成を明記し，オリジナル曲か原作つきの曲か，原作つきの曲の場合，作曲者は誰かなどの情報を明記して，演奏データを送ってください。その際にカセットテープ，CD-Rなどで実際の演奏を録音したものを添付してください（MD，DATなどはご遠慮ください）。ほぼ例外的にX68000内蔵音源のみの場合は特に必要ではありませんが，MIDI使用の場合は録音物の添付をお願いします。

特にゲームミュージックの場合はなにのどの曲かというのを詳しくお願いします。

データの形態はMMLではなく，リアルタイム入力や，シーケンサを使った曲などでもかまいません。使用ツール名などを明記して，SMFだけでなくソースデータファイルも付属してください。

機種固有音源やMIDI使用曲などではデータと一緒に必ず録音テープを添えてください。こちらで同じ環境での再生ができるとは限りません。

いずれの場合も，

〒103-8501
東京都中央区日本橋箱崎町24-1
ソフトバンクパブリッシング（株）
DOS/V magazine編集部
Oh!X制作実行委員会

までお送りください。なお，内容に応じて，

プログラム投稿係
スタッフ募集係
Oh!X LIVE in 係

など，適当に宛先係名を追加しておいてください（厳密な区別はありません）。

または，e-mailで aueki@softbank.co.jp までご連絡ください。

なお，投稿いただいた作品の返却はいたしておりませんので，あらかじめご了承ください。

D3D Immediate Modeを使う 第2回 キューブマッピングと頂点バッファの応用

菊地 功 Kikuchi Isawo

DirectX 7のフィーチャのひとつに6枚の画像を使った環境マッピング，キューブマッピングがある。今回はDirectX 7 SDK収録＆GeForce256導入記念ということで，nVidiaデモ風のキューブマップデモを作成してみよう。

前回は（RMの）記事で，「IMでゲームを作る」なんて大見得切っちゃったんだけど，まあ例によってなにもやっとらんし，今回は特集のほうでRMを使ってゲーム作っちゃったんで，こっちはまたなにか小ネタで行こうかなと。

DirectXも7になって久しいが，X7といえばハードウェアTnL，ハードウェアTnLといえばGeForce256である。現時点ではハードウェアTnLが利用できるのはこの石だけだし，ほかが対応してくるのはもうちょっと先のことだし，Savage2000の例もあるように対応したからといって使いものになるかはわからないし（編注：ハードウェアTnLユニットはポリゴンセットアップユニットと一部共用なので，TnLを無理に使うとセットアップが遅くなるので結局ドライバでTnLをほとんど使わないようにしてるらしい），とりあえずGeForce256が載ったカードは買っておいて損はない（NV15まで我慢するという手もあるが）。筆者はELSAのERAZOR Xを採用してみたが，経済的に余裕があるようなら，カノープスのSPECTRA7400がおすすめ。DDRは……買えるならそれにこしたことはないが，DDRだからといってなにかができるわけではないので，まあ財布と相談して。

ところで，nVidiaのデモをご覧になったことがあるだろうか。TreeMarkはベンチマークも取れることもあって，各誌でよく紹介されることがあるのだが，それではなくて，柔らか系のヤツである（名前忘れた）。マウスでクリックすると，宙に浮いている水銀の玉の表面が波打つデモだ。こいつはハードウェアTnLよりもむしろキューブマッピングのデモなのだが，今回はこれに似たものを自分でも作ってみようと。そんなわけで，今回もまたハードウェアでキューブマッピングに対応したカードでないと，最終的な作品はろくに動かないのでご了承願いたい。



図1　サンプルTexture。動作自体は1999春号のものと同じ

DirectX 7について

その前にDirectX 7。こいつはいままでのバージョンアップのように，インタフェイスのサフィックスが上がったり，メソッドの引数がちょっと変わったり（もちろんメソッドが増えたり）といっただけの，簡単な改変ではなく，結構大掛かりな変更がなされている。X6以前のプログラムをX7に対応させるときにいちばん注意が必要なのは，IDirect3DLightやIDirect3DMaterial，IDirect3DTexture系のインタフェイスが廃止されたこと。従来はこれらのインタフェイスを操作しておいて，IDirect3DViewport3::AddLight()やIDirect3DDevice3::SetLightState()メソッドに渡すなどしていた。これがX7からは，D3DMATERIAL7やD3DLIGHT7構造体に値をセットしておいて，IDirect3DDevice7::SetMaterial()やIDirect3DDevice7::SetLight()メソッドで設定するという，さらにシンプルな方法となった。また，テクスチャはサーフェスと統合され，IDirectDrawSurface7インタフェイスで一括管理される（テクスチャはサーフェス作成時にフラグとしてDDSD_TEXTURESTAGEを，VapsとしてDDSCAPS_TEXTUREを取る）。IDirect3DViewport3インタフェイスも廃止され，D3DVIEWPORT7構造体に値をセットしたあと，IDirect3DDevice7::SetViewport()メソッドを使用する。こういった改変により，IMのインタフェイスの数は，IDirect3D7，IDirect3DDevice7，IDirect3DVertexBuffer7の3つになってしまった。なお，Direct3D IMに限らず，DirectX 7ではサフィックスのつくインタフェイス名を，混乱を避けるために一部を除いて"7"で統一したようだ。一部ってのはDirectX 7で特に機能が変わらなかった部分で，RMもそれに含まれる。いよいよもってRMは見捨てられそうな雰囲気だ。それよりもVBへの対応に力が入れられている。

そんなわけで，1999春号のサンプルTexture（DirectX 6）を，DirectX 7で書き直したものも付録CD-ROMに収録しておいた。1999春号が手元にある人は，2つのソースを見比べてどれくらい変わったかを確認してもらいたい。EnumDeviceも当然DirectX 7に対応させているので，ハードウェアTnLがあれば，デバイスに列挙されるようになっている。

波のシミュレーション

いきなり球面というのは難しいので，まずは平面で考えてみよう。ある点で振動が発生したとして，その点の運動を，

$$f(t) = k1^t \times \sin(\omega \times t)$$

としよう。tは時間，k1は時間に対する減衰定数，ωは角速度だ。この点から距離x離れた点では，

$f(t,x) = k2^x \times f(t-x/v,0)$

ただし$t \geqq x/v$

となる。k2は距離に対する減衰定数，vは波の速度である。この2つの式から，任意の点xにおける時刻tの水面の高さは，

$f(t,x) = k1^t \times k2^x \times \sin(\omega \times (t-x/v))$

という式が導ける。ここまではどうということもない。面倒なのは法線だ。とりあえずこいつをxで微分してみる。

$f(t,x)/dx = k1^t \times (\log k2 \times k2^x \times \sin(\omega \times (t-x/v)) + k2^x \times \cos(\omega \times (t-x/v)) \times -\omega/v)$

$= k1^t \times k2^x \times (\log k2 \times \sin(\omega \times (t-x/v)) - \omega/v \times \cos(\omega \times (t-x/v))$

でいいんだよなぁ，多分。ここで，k2は減衰定数であるので1よりは小さいが，まあ1に十分近いとしてしまえば，$\log k2 \fallingdotseq 0$となって，sin成分は省ける。法線は陰影となって現れるが，そんなに真面目に計算しなくても，そんなものはそれっぽけりゃわからない。そんなわけで，法線は$\cos(\omega \times (t-x/v))$に適当な係数をかけてやった値を基準に算出することにする。

この理屈を基にして，プログラミングを行う。まずは1枚の板。これを波打たせるわけだから，無数のポリゴンを敷き詰めて作る。今回は頂点バッファを使ってみよう。頂点バッファとは，その名のとおり頂点のバッファである。機能的に単なるメモリ配列となにが違うかというと……なにが違うんだろう。ヘルプには「シーン内にモデルを必要な数だけ再変換しないでレンダリングできる」とあるが，別にメモリ配列だって，変換された頂点を含めばいいのでは？　とりあえず書き込みだけでいいならば，頂点バッファの作成時にD3DVBCAPS_WRITEONLYフラグとかで最適化すれば，メモリ配列よりは速くなるはずではあるが。

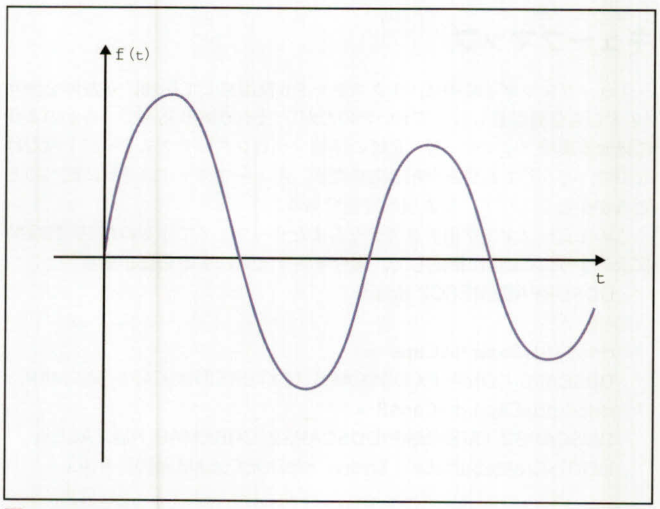


図2　震源の運動

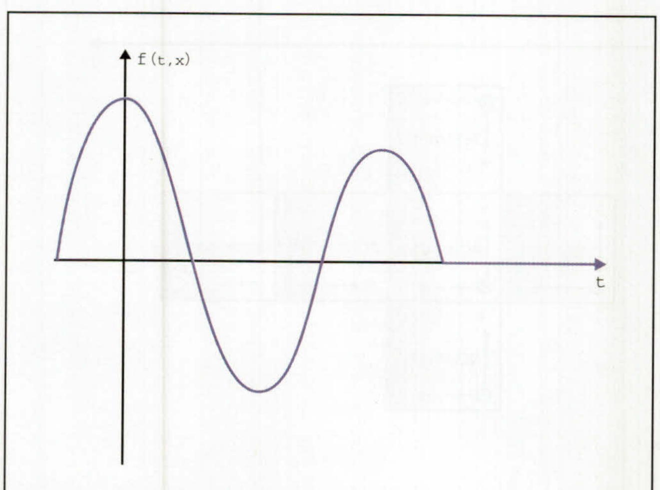


図3　周囲の変移

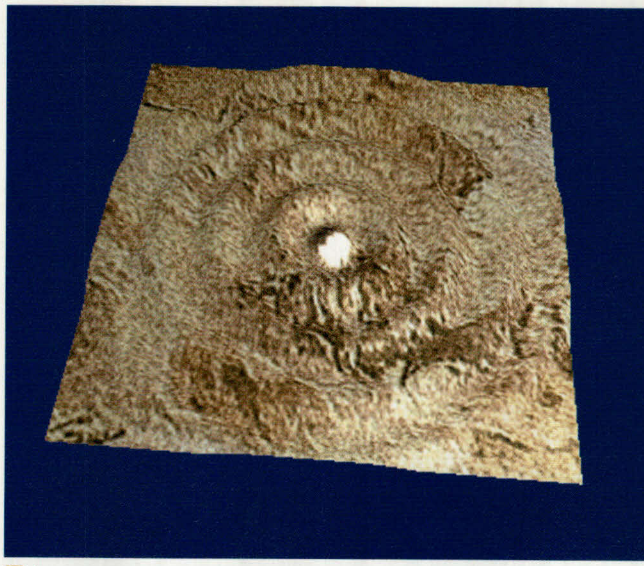


図4　サンプルWave。中心で起きた波紋が減衰しながら周囲に広がる

頂点バッファは，先ほど少し出てきたが，IDirect3DVertexBuffer7というインタフェイスを使う。このインスタンスの作成には，D3DVERTEXBUFFERDESC構造体が必要だ。

```
D3DVERTEXBUFFERDESC vbdesc;
vbdesc.dwSize = sizeof(D3DVERTEXBUFFERDESC);
vbdesc.dwCaps = D3DVBCAPS_WRITEONLY;
vbdesc.dwFVF = D3DFVF_VERTEX;
vbdesc.dwNumVertices = NUM_VERTICES;
```

D3DVBCAPS_WRITEONLYは先ほど述べたとおり。D3DFVF_VERTEXは，この頂点バッファではD3DVERTEX構造体と同じメンバを持ちますよ，ということ。つまり，変換前の頂点座標，法線，それにテクスチャ座標だ。NUM_VERTICESは頂点座標で，文字定数で定義しておくといいだろう。また，HELならばdwCapsにD3DVBCAPS_SYSTEMMEMORYフラグも設定しておく。

```
lpD3D->CreateVertexBuffer( &vbdesc, &lpD3DVBuf, 0L );
```

としてIDirect3DVertexBuffer7のインスタンスを作ったら（lpD3DはIDirect3D7のインスタンス，lpD3DVBufはIDirect3DVertexBuffer7のインスタンスを受けるポインタ），

```
lpD3DVBuf->Lock( DDLOCK_WAIT, (void**)&pFlagVertices,
NULL );
```

としてロックしたあと（pFlagVerticesは頂点配列を受けるポインタ），頂点などの書き込みを行い，

```
lpD3DVBuf->Unlock();
```

でロックを解除する。レンダリングには，IDirect3DDevice7::DrawPrimitiveVB()あるいはIDirect3DDevice7::DrawIndexedPrimitiveVB()メソッドを使う。前者は，頂点座標の順にポリゴンを作る場合に使い，後者は別途ポリゴンを構成する頂点のインデックス配列を用意する場合に使う。複数のポリゴンがひとつの頂点を共有することがある場合には，後者になるだろう。RMのフェースデータに相当する。

板の大きさは64×64とした。頂点数は65×65=4225，IMの場合はポリゴンは三角形に限られるので，三角形分割してポリゴン数は64×64×2＝8192枚となる。それを毎フレーム，上で求めた式を基にして座標，法線を計算，レンダリングする（サンプルWave，図4）。波は一定時間ごとに中心

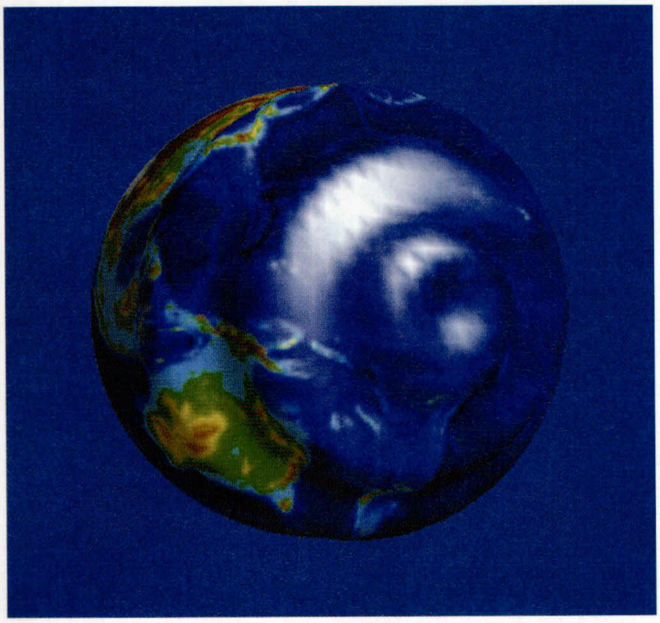


図5　サンプルBubble。波打つ地球

で起こるようにしてあるが，この程度の処理ならば，それほど強力なCPUやビデオカードでなくても，十分滑らかに動くはずだ。

球面に拡張

これを球面にすること自体は難しくないだろう。板の大きさを128×64にし，それをメルカトル図法的に球にする。ただし，その場合，震源からの距離の算出方法は異なってくる。元の板の座標からいろいろと計算するよりは，震源との角度を求めてしまったほうが簡単だろう。これには外積を用いる。内積でも角度を求めることはできるが，外積で得られるベクトルは，法線を求める際の回転軸としても利用できるからだ(外積については，特集のSION2000の記事を参照のこと)。外積を求める関数は，IMにも用意されている。

```
_D3DVECTOR CrossProduct (const _D3DVECTOR& v1, const
    _D3DVECTOR& v2);
```

実際にはこれはインライン変換される。この関数を利用するには，(デフォルトでは)C:¥mssdk¥samples¥Multimedia¥D3DIM¥includeにあるD3DUtil.hをインクルードし，C:¥mssdk¥samples¥Multimedia¥D3DIM¥libのd3dframe.libをリンクする必要がある。これらのフォルダはVisualStudio側のインクルードとライブラリフォルダに登録しておくといいだろう。ただし，これを毎フレーム行うのはちょっと重そうなので，あらかじめ頂点数分のバッファを確保しておき，そこに震源からの距離(角度)と軸ベクトルを保存しておく。震源が常に同じ位置という前提だ。

さて，法線ベクトルであるが，まずは平面であると仮定して求める。その後，保存しておいた角度と軸ベクトルで回転を行う。ただ，なぜかIMにはベクトルを回転させるヘルパー関数がない。そこで，ちょっと気持ち悪いが，RMのD3DRMVectorRotate()を使うことにした。単なるヘルパー関数なので，特にRMのインタフェイスの初期化などを行わなくても，d3drm.hのインクルードとd3drm.libのリンクだけで使えるようになる。

そんなことでできたのがサンプルBubbleだ(図5)。先ほどと同じように，定期的に波紋が走る。丸い物体が波打った場合，裏まで波紋が伝わったあと，反対側からまた返ってくるはずであるが，このサンプルは裏側までしかシミュレートはしていない。また，長方形を球にする際，北極点と南極点は，それぞれひとつの頂点にまとめれば頂点数もポリゴン数も稼げるが，面倒なのでそのままにしてある。

キューブマップ

キューブマップってのは，テクスチャを6枚用意して周囲に立方体を作り(必ずしも全面用意しなくてもいいのだが)，それが映り込んでいるかのように見せる環境マップの一種。RMの環境マッピングはテクスチャが1枚だけなので，どうしても歪みが起きるのだが，キューブマップはそれに比べると歪みが少なく，リアルな表現が可能である。

このキューブマップ用テクスチャもまたサーフェスで作るのだが，作成時に次のようにCapsに指定して，複合サーフェスにする必要がある。

```
DDSURFACEDESC2 ddsd;
        ︙
ddsd.ddsCaps.dwCaps  =
DDSCAPS_COMPLEX|DDSCAPS_TEXTURE|DDSCAPS_MIPMAP;
ddsd.ddsCaps.dwCaps2 =
DDSCAPS2_CUBEMAP|DDSCAPS2_CUBEMAP_ALLFACES;
lpDD->CreateSurface ( &ddsd, &lpDDSCubeMap[0], NULL ) ;
```

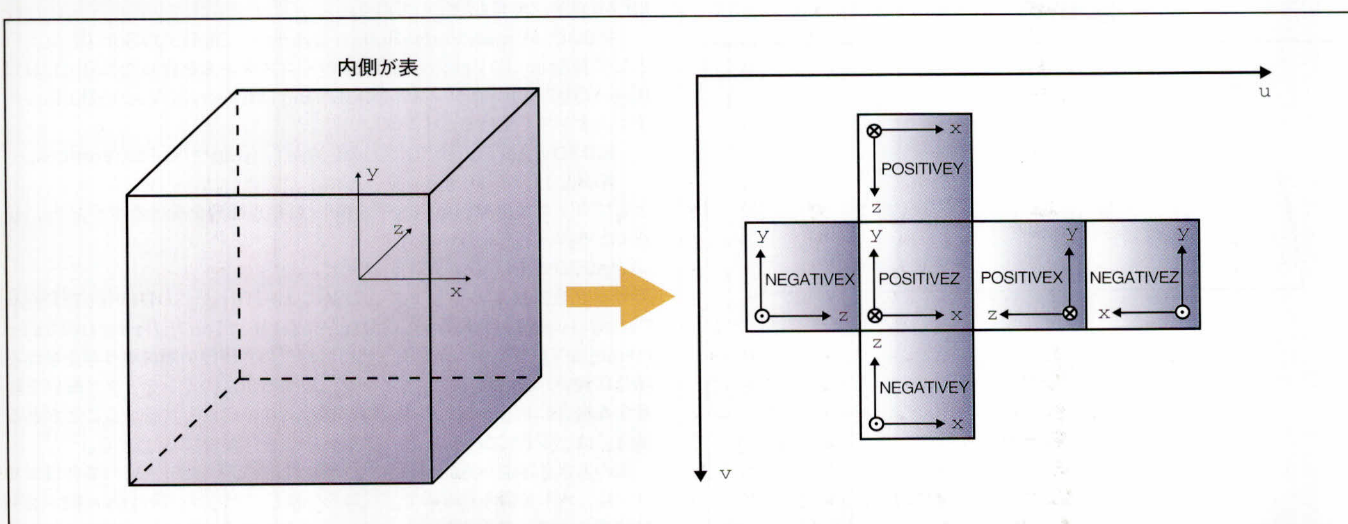


図6　キューブマップテクスチャのフラグとuv座標軸

これが6枚のうちの１枚目，Xの正方向に貼り付けるサーフェスとなる。残りの5枚は，Caps2にDDSCAPS2_CUBEMAP_NEGATIVEX，DDSCAPS2_CUBEMAP_POSITIVEY，DDSCAPS2_CUBEMAP_NEGATIVEY……といったフラグを指定し，

```
SCubeMap[0]->GetAttachedSurface ( &ddsd.ddsCaps,
&lpDDSCubeMap[n] ) ;
```

として，複合サーフェスにアタッチする。こうすれば，

```
lpD3DDev->SetTexture ( 0, lpDDSCubeMap[0] ) ;
```

として1枚目のテクスチャをテクスチャステージにセットしただけで，自動的にほかの5枚もキューブマップに利用される。問題となるのは頂点バッファのほうだ。キューブマップは単にテクスチャを6枚放り込めば全自動ですべてやってくれるというものではない。たとえば普通にポリゴンにテクスチャを貼ろうと思ったら，それぞれの頂点にテクスチャ座標が必要だろう。キューブマップの場合も同様に，各頂点にテクスチャのどの部分を貼るかを指定する必要がある。ただし，キューブマップの場合は，テクスチャのuv座標ではなく，立方体の中心から見たベクトルで指定する。このベクトルから，システムは6枚のうちのどのテクスチャの，どの部分に当たるかを計算して貼り付けることになる。ここに，任意の頂点から視点までのベクトルと，法線ベクトルに対して対称な反射ベクトルをセットすれば，あたかも周囲が映り込んでいるように見えるというわけだ(周囲は無限遠にあるものと仮定する)。

そうなると，頂点バッファはフォーマットの変更が必要になる。先ほどは頂点座標，法線ベクトル，テクスチャuv座標を含んでいたが，今度は3次元のテクスチャ座標が必要になる。これはひとつの定数として用意されていないので，複数の文字定数のコンビネーションで表現する。頂点座標はD3DFVF_XYZ，法線座標はD3DFVF_NORMALである。テクスチャ座標自体はひとつでいいので，D3DFVF_TEX1と，最初のテクスチャ座標は3次元なんだよ，ということで，D3DFVF_TEXCOORDSIZE3(0)マクロを利用する。これらをorでつないだものをD3DVERTEXBUFFERDESC構造体のdwFVFメンバに設定する。これで頂点バッファの各頂点に，3次元のテクスチャ座標を設定できるようになった。この座標に，毎フレーム反射ベクトルを設定することになる。

ここまでがサンプルBubble2だ(図7)。キューブマップに使っているテクスチャがわかるように，周囲にも実際に立方体を設置している。かなり自然に周囲が映り込んでいるように見えるだろう。

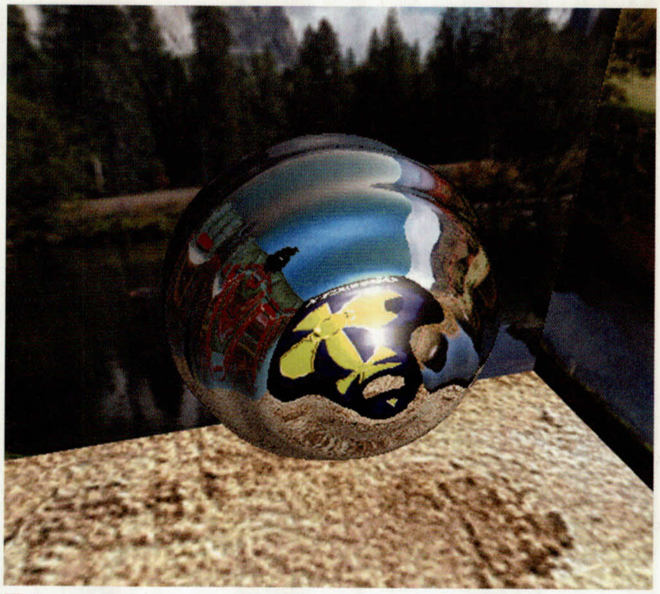


図7　サンプルBubble2。浮遊する水銀球

オブジェクトの映り込み

実際のシーンでは，キューブマップとして用意した画像が映り込むだけでは，あまり役には立たない。やはり周囲のオブジェクトも映り込まなくては。環境マッピングと2段レンダリングによる周囲のオブジェクトの映り込みは，1999春号のRMでもやったが，あのときは歪みが大きかったというだけでなく，一部のビデオチップではレンダリングしたサーフェスをテクスチャとして利用できないため，正常に動作しないという問題があった(いや，正常に動作するビデオチップのほうが少なかったか)。しかし，DirectX 7からは，冒頭でも述べたようにテクスチャとサーフェスが統合されており，2段レンダリングも正式にサポートされている。

その方法は，想像のとおりだ。ビューを中心に置き，そこから6方向に向けてレンダリングを行う。描画先はキューブマップサーフェスだ。このため，キューブマップサーフェスは作成時に，CapsにDDSCAPS_3DDEVICEフラグを指定しておく必要がある。最後にそのキューブマップサーフェスを使って，シーンを描画する。ここは先ほどと同じだ。むー，それ以外特に説明する部分はないなぁ。強いていえば，レンダリングするターゲットの変更方法だが，この辺はSDKのサンプルから引っ張ってきたものなので，まあ各自で確認してもらいたい。

で，完成(サンプルBubble3・図8)。周囲を回っているトラとティーポットが映り込んでいるのがおわかりいただけるだろうか。しかし，やはりキューブマップは無限遠が前提なので，オブジェクトの映り込みが小さめで，やや不自然になってしまっている。キューブマップサーフェスを作る際のビューの位置を，ちょっと外に寄せるなどの小細工をすれば，この辺りはもう少し自然になったかもしれない。ま，その辺は各自で試してもらいたい。ただし，その場合にはフロントクリップに要注意だ。

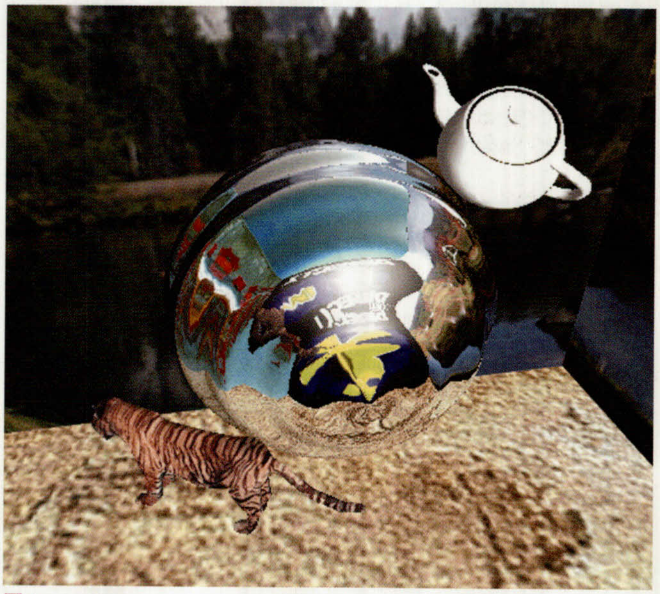


図8　サンプルBubble3。トラとティーポットの映り込みがやや小さいか

さて，このサンプルをERAZOR Xで試したところ，単なるHALデバイスよりも，ハードウェアTnLデバイスのほうが遅くなってしまった(CPUはPentium III/558MHz)。どうやらビデオは割と一杯一杯で，CPUが遊んでいるらしい。それほどキューブマッピングは重いということだろうか。ま，1フレーム描くために，7回レンダリングを行っているのだから，そりゃそーか。nVidiaのデモのように，マウスでクリックして波をたくさん増やすことができれば，それなりにCPUの仕事も増えるんだが。ま，こっちはオブジェクトの映り込みまでやったということで，ご容赦願いたい。

かなりの勢いで細かいところをはしょって説明してしまったが，RMならいざ知らず，IMを使う人間ならこれくらいは理解できると信じたい。お気楽ではIMはマスターできんぞ。精進するように(>筆者)。さて次回は……DirectX 8が出てたりして。

存在感のある効果音の作り方 ～ステレオ音場効果の基礎～

舩本昇竜/満開製作所　Funamoto Shouryu

「サウンド処理」というと，音楽演奏などが思い浮かびますが，発音するデータにさまざまなエフェクトを施すことで音声をよりリアルに再現することも可能です。デジタル信号処理による音場効果の作り方について考えてみましょう。

アーケード版『沙羅蔓蛇』あたりから，ゲームもステレオ時代に本格突入。というのが一般教養かな，などというタワゴトはさておき，そのころから数多くの，素晴らしい「ステレオ曲」が生まれてきたのは，紛れもない事実です。ソレに対して，素晴らしい「ステレオ（効果）音」というのは，案外，お耳にかかれないのではないでしょうか（CAPCOMゲームの「Qサウンド」などは別として）。

それでも，ここ最近，かなり凝った3D効果音，とでもいいますか，「ゲームという空間」を演出してくれる「効果的な効果音」をよく耳にするようになりました。では，そのような，「音から空間を感じる」，もしくは「空間を感じさせる音」とは，どのようなモノで，どうやって生み出されるのでしょうか？

いきなり解答を出してしまうと，そういった音を作るには「音場処理」を効果音に加えてやればいいのです。では，その「音場処理」とは？　ってことになるのですが，それについては，これからゆっくり説明していきましょう。

左耳で聞く音，右耳で聞く音

人間には，左右，あわせて2つの耳があります。これらの耳は，意外に思われるかもしれませんが，日常生活において，左右の耳が「同じタイミングで，まったく同じ音を聞く」ことは，ほとんどありません[※1]。そして，我々は，左右の耳で聞いたそれぞれの音の微妙な違いをもとに，音の定位，つまり「どの方向から音がやってくるのか」を判別しています。それも，ごく自然に。

では，耳はどのようにして判別するか，そしてどのようなことを判別の対象とするかなどについての重要なポイントを3つ解説しましょう。

※1：「密封型ヘッドフォンをつけ，左右同じ音を鳴らす」というような，特定条件下にいる場合は別です。

音が乗り越えてくる距離

「千里先で10円玉が落ちた音を聞くことができる」などという台詞を，たまに聞くことがあると思いますが，我々の日常生活の場においては，本当にそんなことができる人は，まずいません。というのも，人間の耳のよし悪し以前に，音は，遠くに行けば行くほどだんだん弱く（聞こえないほどの大きさに）なり，そして消えていきます。このような現象を音の「減衰」といい，対象となる音との距離が開けば開くほど，大きく減衰します。

さて，図1を見てください。図は，人間から見て，正面やや右側に音源があり，左右の耳が聞くであろう音の道筋をそれぞれ赤と青の線で表してあります。ここで，注目してもらいたいのが，赤と青の線の長さに，わずかであるものの，違いがあるというコト。音源が正面やや右側にあることから，赤線（音源と左耳との間の距離）は青線（音源と右耳との間の距離）より，長くなっています。つまり，左耳に届く音は，距離が長い分右耳に届く音より減衰するため，右耳で聞こえる音より小さくなります。

なお，距離と音の減衰の間には，「距離が2倍になると，音の大きさは1/4になる」（距離の2乗に反比例する）というルール[※2]があります。ではここで，図1のような状況の場合，どの程度音が減衰するのか計算してみましょう。図2を見ながら，音源から左右の耳までの距離を，ピタゴラスの定理を用いて算出してください。音源から右耳までの距離を1（基準）として考えた場合，音源から左耳までの距離は1.06となります。この結果に，上記のルールを当てはめると，左耳で聞こえる音は，右耳で聞こえるソレの89%の大きさになることがわかります。ちなみに，基準の取り方は環境によっていろいろ差が出ますが，間違いないのは，図のような場合，多少の差はあったとしても，必ず，右耳の方が大きな音を聞いているのです。

※2：これを「拡散減衰」といいます

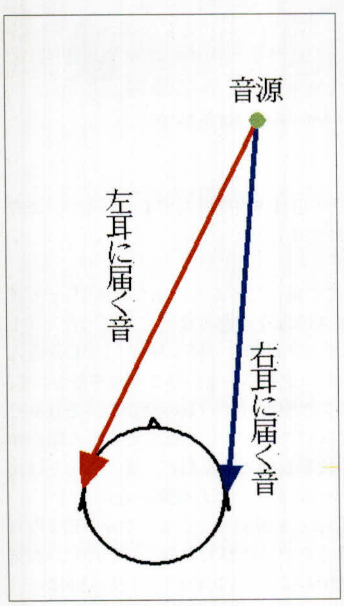


図1　音源から音が耳に到達するまで

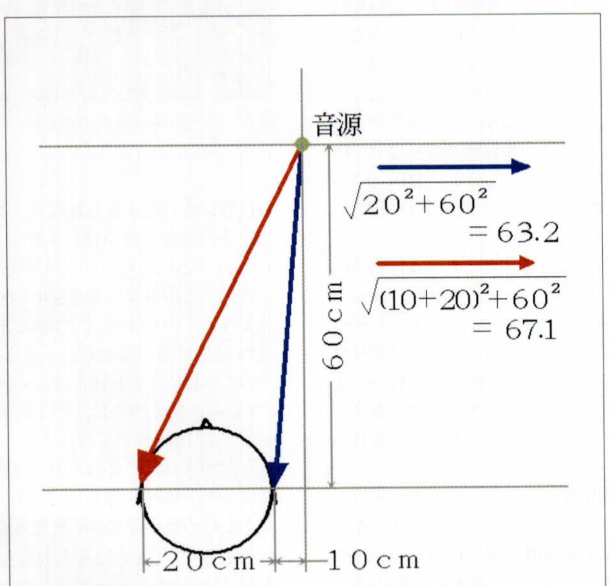


図2　音源からの距離の違い

マッハの速度

音速という言葉があるように，音には「空気中（媒質）を伝わる速度」があります。つまり，どんなに「スグ聞こえた」と思う音も，空気中を伝わる時間があり，決して，0秒ではないのです。音速は「マッハ1」とも呼ばれ，その速度は，秒速340m/秒[※3]。「マッハの速度のジェット戦闘機」と聞くと速そうですが，

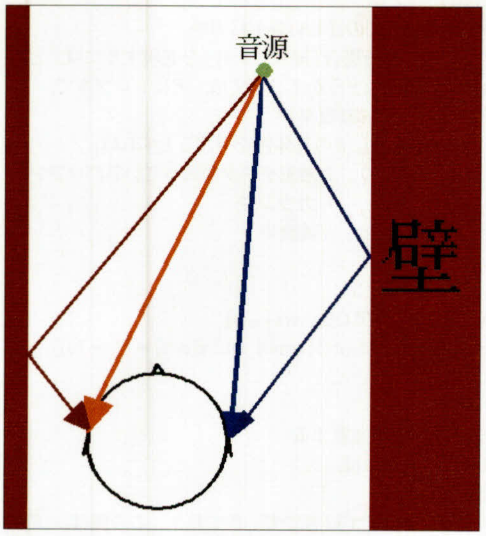


図3　壁に当たってから耳に届く音



図4　壁に当たった音

「マッハの速度の音」と聞くと，なんだか速いのか遅いのか，その感覚がピンときません。ですが，ショージキなトコロ，実は，音って，結構遅いモノだったりするのです。

再び，図2をご覧ください。この場合，音源から発信された音が，実際に右耳に到達するまでには，

0.671（m）÷340（m/秒）＝0.001859（秒）

かかることになります。つまり，「63cm進むのに1m秒（1/1000秒）以上かかる」のです。こう書くと，妙に遅く感じませんか？　同じように，音源から発信された音が，今度は左耳に到達するまでの時間も計算してみましょう。

0.671（m）÷340（m/秒）＝0.001973（秒）

ちょっとだけですが，違いますね。これは，どういうことかというと，音源から発信された音は，「異なったタイミングで耳に到達する」ことを表しているのです。たった0.000114秒というタイムラグ。「たかが1万分の1秒」と，無視ししたい気持ちもわからなくはないのですが，このタイムラグの間に，「100MIPSのプロセッサがいくつの命令を実行できるか？」と考えると，これは，決して無視できない時間だったりするのです（ちょっと例えがヘンかな？）。

なにより，このごくわずかなタイムラグ。頭で理解できるかはさておき，実際，我々の耳は，その差を感覚として感じることができるのです※4。

※3：厳密には「気温15℃における空気中の速度」なのですが，音速といえばこの値を使うのが普通になっています。
※4：もちろん個人差はあります。

ハネッカエリな音

これまで見てきたように，我々が聞いている音は，減衰し，時間をかけて，ようやく耳に到達します。もちろん，耳に到達するまでに，音そのものが小さくなりすぎてしまうと，「音が聞こえる」ことすら知覚されません。

ところで，我々は，図1で示すような，音源からまっすぐ届く音だけを聞いているわけではありません。図3をご覧ください。かなり極端な例ではありますが，我々は，まっすぐ耳に届く音のほかに「壁に当たり，反射してから耳に届く音」も聞いているのです。

基本的に，壁に反射してから耳に到達する音は，直接耳まで届く音に比べ，さらに遅れ，さらに減衰してから耳に届くため，直感的な音というより「音の余韻」「響き」として知覚されます。壁に当たってからという遠回りをして到達が遅れる，というのは直感的に理解できると思いますので，ここでは，壁に当たることで音がどのように減衰するかについての仕組みを見てみましょう。図4をご覧ください。便宜上，「線の太さ＝音の大きさ」と考えてください。まず，音※5が壁に当たります。壁に当たったひとつの音は，まず，「壁に当たり跳ね返る音」と「壁の中に入る音」に二分されます。ここで重要なのが，

壁に当たった音＞跳ね返った音

そして，

壁に当たった音のエネルギー＝跳ね返った音のエネルギー＋壁の中に入った音のエネルギー

ということです。さて，跳ね返った音は，無事，耳まで届いたとして，壁の中に入った音はどうなるのでしょうか？　壁の材質にもよるのですが，たいていの場合，壁（の材質）に吸収され音としての存在が消滅してしまいます（これを「吸音」といいます）。それでも，しぶとく生き残った音は，なんと壁を通り越し，壁の向こう側で聞こえる「通過音」として生まれ変わり，第二の人生を歩み始めるのです。

※5：壁に当たるまでの空気中で拡散減衰していることをお忘れなく。

三種の神器で音場処理に挑め

これまでに述べた音に関する3つの項目，「減衰」「遅延」「反射」をある程度抑えることができれば，市販されているDSPアンプなどで実現しているような「音場処理」に「少し」は近づけるようになります。あくまでも「少し」



ではありますが，経験的に，いろいろと遊べ，結構楽しいモノです。

ということで，図5を見てください。ごく簡単かつ最低ラインのモノながらも「コンサートホール」の音場シミュレーション（苦しいのはガマンガマン）になっていることに気づかれたでしょうか？　実際のコンサートホールを設計する場合は，建物の形状にともなう，もっと複雑で数の多い壁や天井，座席や通路はもちろんのこと，空調システムなども考慮されているハズです。あとは，近くの地下鉄※6の状況を把握するのも重要かと。

余談はさておき，音源の「左右」位置感覚の違いを感じさせることができるのは，基本的に，直接耳に届く音だけです。実際，ある程度（3～5方向程度）の音源の「左右」位置感覚の違いを出すだけであれば，「直接耳に届く音の遅延」※7を考慮するだけでできてしまうものです。逆に，直接耳に届く音だけでも，減衰と遅延をキチンと考慮することで，前方「10時半の方向」と「11時の方向」という微妙なニュアンスの違いを出すこともできます。

では，壁に当たってから耳に届いた音はなんの役に立つのでしょうか？もっとも意味のある答えは，「音の広がり」をはじめとする，「空間」に関するニュアンスの違いを出せることです。実際のコンサートホールや教会などを例に考えるとわかりやすいのですが，壁の材質が違う，すなわち壁に音が当たったときの反射率などが変わると，特に音の「響き」の部分に関するニュアンスに変化が起こります。再び図5を見てください。一度壁で跳ね返り，減衰し，若干遅れて耳に入る音が何種類か存在します。このとき，我々はそれらを「音」としてではなく「響き」と認識し，その響き具合でその場の「空気感」を経験的な学習結果から導き出すのです。

・狭い空間で音が鳴っている場合，音と響きの感覚が短く，広い空間の場合は音と響きの感覚が長い
・音がよく跳ね返る（吸音率の低い＝減衰しにくい）壁の中では，結構長い間響き続ける

と，いったことを，理論的に考えず，経験と照らしあわせることで「まるでコンサートホールにいるようだ」というような感覚を得るのです。

※6：レナード・バーンスタイン（私がもっとも好きだった指揮者）が地下鉄を止めてオーケストラを指揮した。という伝説が録音史に残っています。
※7：よく「左右の位相を少しずらす」と表現されます。ちなみに，スピーカーケーブルの＋－を片方だけ間違えると，「左右の位相が180°ずれた状態」になり，「妙なステレオ感」が得られます。あまりうれしくありませんが。

音場をプログラミングする

ここまでの説明が，結構長くなってしまったので，実際のプログラミングについては，駆け足気味でまいりましょう。ということで，重要な項目だけれども，符号付き16ビットPCM（Pulse Code Modulation）方式データの説明とサンプリング定理についての詳しい説明は省略。

まず，波形データを相手にした場合「減衰した音」を表現するには，どのような処理を加えればよいのでしょうか？　答えは，実にシンプルで，

減衰した音＝元の音×（1－減衰率）

とこれだけでOK。プログラムとしての具体例を挙げるとすれば，

```
short   wave_data[44100];   //波形データが入っているバッファ
int  wn;                    //カウンタ
float   lv=0.2;             //減衰率
main()
{
        for( wn=0; wn<44100; wn++ ){
        wave_data[wn]=(short)(wave_data[wn]＊(1 − lv));
        }
}
```

・wave_data［］の波形データを減衰する

という具合になります。個人的には，

増幅率＝（1－減衰率）

という考え方がイメージしやすくて好きです。さておき，この例は，「符号付き16ビットPCM」のデータであれば，どのようなサンプリング周波数のデータでも適用可能です。それにしても，見たマンマ「パラメータ命」型プログラムですね。余談ながらも，このプログラムにファイルの入出力や減衰率を設定できるようにすれば，あっという間に「簡易デジタルボリューム」に変身するのですが，気がつきましたか？

では，次に「遅れてやってきた音」を表現するには，どのような処理を加えればよいのでしょうか？　これも原理がわかれば，とても，簡単なことです。

```
short   wave_data[44100];   //波形データが入っているバッファ
short   master_data[1024 ＊ 1024];   //最終出力バッファ
int     wn,dc;   //カウンタ／ディレイカウント値
float   delay=0.001858;   //遅れ時間(sec)
#define   SMP_RATE   (44100)   //サンプリングレート(44.1kHz)
main()
{
      dc=(int)(delay ＊ SMP_RATE);//ディレイカウント値の算出
      for( wn=0; wn<1024 ＊ 1024; wn++ ){
          if( dc<=wn && wn<(dc+44100) ){
                master_data[wn]=wave_data[wn-dc];
                //ディレイカウント分遅れたデータを出力
          }
          else{
```

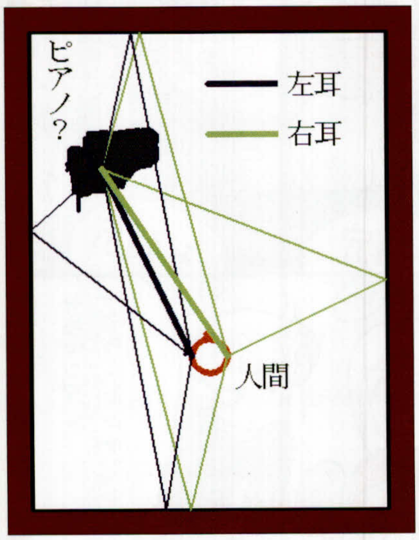


図5

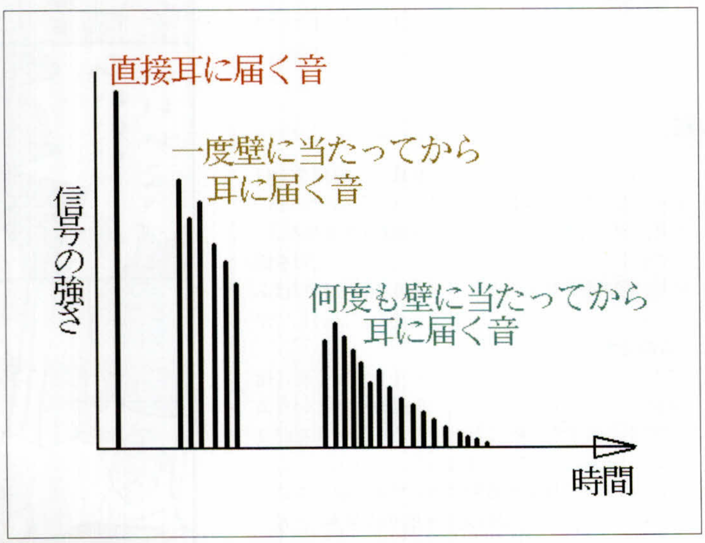


図6

```
            master_data[wn]=0;
        }
    }
}
```

・wave_data[]のデータをdelay秒遅らせて，master_data[]に格納

ご丁寧なチェックをしているので，少し長くなりましたが，キーポイントは，

```
master_data[wn]=wave_data[wn-dc];
```

下線部のように，配列要素数を使い「遅れ」を表現していることです。たとえば，44.1kHzの波形データの正体は，1秒あたり44100サンプリング(=88200byte)分の解像度を持つデータであり，上記の例のように0.001858秒遅れるということは，81(≒44100×0.001858)サンプリング遅れることを意味します。これが，32kHzの波形データである場合，1秒あたり32000個(＝64000バイト)分の解像度を持つデータなので，59(≒44100*0.001858)サンプリング遅れることになります。

ここまでは，比較的簡単だったと思います。では，残りの「『直接耳に入る音』と『壁に当たってから耳に入る音』を同時に感じる」とは，処理的にどういったモノなのでしょうか？　ワザと難しさを匂わせるマエフリをしましたが，実はというか，やっぱり，これも簡単なのです。

```
short  main_data[44100];  //耳に直接届く波形データが入っているバッファ
short  sub_data[44100];  //壁に当たってから届く波形データが入っているバッファ
                         //↑どちらも遅延・減衰済みと仮定
short        master_data[44100];        //最終出力バッファ
int   wn;                              //カウンタ
main()
{
      for( wn=0; wn<44100; wn++ ){
            master_data[wn]=main_data[wn]+sub_data[wn];
      }
}
```

・main_data[]のデータとsub_data[]のデータを同時に聞くイメージ

これだけです。意外に感じる人も多いと思いますが，最大のキーポイントは，

```
master_data[wn]=main_data[wn]+sub_data[wn];
```

下線部で示すように，同時に音を感じるという処理は「ただの足し算」で表すことができるのです。知識の多い人ほどかえって，「2つの波形をFFTし，両方のスペクトルを足し合わせたうえで，iFFTするのでは？」と思われているかもしれませんが，実際には，ただの足し算で十分なのです。

乗算および加算命令が支配する世界へ

最後に，拙作の音場処理付き波形再生プログラムのキモの部分を抜き出し，実際に眺めてみましょう。

「←←？」の注釈のある行に注目してください。原理的な順で流れを追うと，まず，「←←2」の行で，多少ゴチャついてますが，遅延に関する処理(引き算してますよね)を行い，「←←3」の行で，減衰に関する処理(掛け算してますよね)を行っています。そして「←←1」の行により，同時に耳に入る音の本数分だけループ処理(足し算の回数を支配していますよね)を行います。

より高度な音場処理に向けて

駆け足で音場処理のイロハについて触れてみましたが，経験的にいって，音の世界は深いです。ホント，嫌になるくらい深かったりします。たとえば，「なぜバイオリンは，あんな音が鳴るのか？」について考えた場合，なんと，弦はもちろんのこと，「胴体内での音場効果」が大きなポイントになっているのですから，オドロキです。

リスト

```
#define VDSP__BUFLEN    (16*1024)       //ストリーミングバッファの個数
//↑最大バッファ番号より多いこと

unsigned short VDSP__nowpos;            //現在位置
int        VDSP__buf[VDSP__BUFLEN];     //ストリーミングバッファ(stereoで格納)
short      VDSP__offset[]={             //アンプに通すバッファ番号
   0,  838,    2028,   2029,   2293,    -1         //hall
};

short      VDSP__offset_list[4][6]={    //アンプに通すバッファ番号のラインアップ
   0,  838,    2028,   2029,   2293,    -1,        //hall
   0,  63,     100,    510,    1051,    -1,        //disco
   0,  354,    1701,   8156,   12276,   -1,        //stadium
   0,  277,    277,    373,    405,     -1         //jazz
};

short  VDSP__amp[]={                    //増幅率リスト
20070, 8461,   5970,   5472,   5046,    -1         //1,15,0 signed:hall
};

short  VDSP__amp_list[4][6]={           //増幅率リスト
   20070,  8461,   5970,   5472,   5046,        -1,        //hall
   9842,   7047,   7022,   6764,   6424,        -1,        //disco
   15104,  8809,   7908,   4029,   2103,        -1,        //stadium
   7994,   8687,   8688,   6792,   13101,       -1         //jazz
};


int    VDSP__Set(in_data)
int    in_data;        //ステレオデータペア(short+short)
{
   int suml,sumr;
   int res;
   short   l,r;
   short   an;
   short   offset;

/*演算なし ストリーミングデバッグ*/
// res=in_data; goto quick_exit;

   VDSP__buf[VDSP__nowpos]=in_data;  //入力データのセット

   suml=0;
   sumr=0;
   an=0;
   while( VDSP__offset[an]!=-1 ){                          //←←1
      offset=(VDSP__nowpos-VDSP__offset[an]);              //←←2
      if( offset<0 ){ offset+=VDSP__BUFLEN; }
      l=*(short *) (&VDSP__buf[offset]);
      r=*(short *) ((void *) (&VDSP__buf[offset])+2);
      suml+=(int) (l*VDSP__amp[an])>>15;                   //←←3
      sumr+=(int) (r*VDSP__amp[an])>>15;                   //←←3
      an++;
   }
   l=COMP__Set(suml);                  //実は利用頻度が極端に低いとのウワサ
   r=COMP__Set(sumr);                  //食わせる*.S44次第ではあるのだが
// l=(short)suml;
// r=(short)sumr;
   suml=(int) l<<16;suml&=0xffff0000; //このへん勝手に最適化される
   sumr=(int) r;   sumr&=0x0000ffff;
   res=(suml|sumr);
   VDSP__nowpos++;
   if( VDSP__nowpos>=VDSP__BUFLEN ){ VDSP__nowpos=0; }

//quick_exit:
   return(res);
}
```

余談ついでといいますか，結構重要な，音についての雑学をアレコレ。

・空気の温度によって，音速は変化し，暖かい方が速い
・空気中の音速と水中の音速は，速さが異なり，水中の方が速い

ついでに，一歩間違えばギャグな事実を紹介しましょう。まず，以上のことをふまえた上で「ディスコでフィーバー」な音場処理を行う場合は，

「若者の熱気＝温度の上昇」
「若者の汗＝湿度(空気中の水分率)の上昇」

のため，音速をかなり高めに，減衰率を若干低めに設定します(やりすぎると「サウナ」になるので注意！)。うむむ，やっぱり，音の奥は深いですね。

参考文献

DSP-R793取扱説明書，ヤマハ
良い録音を行うためのレコーディング・ミキサー心得帖，荻野宜邦，リットーミュージック
図解雑学　音のしくみ，中村健太郎，ナツメ社
DSPLAY.X(月刊電脳倶楽部第136号)，舩本昇竜，満開製作所

言語処理プログラムを作る　第2回
変数の管理と中間コード

石上 達也　Ishigami Tatsuya

ポータルブルな言語システムを作ろうという連載ですが，今回は変数を扱い，さらに中間コードの処理にまで進みます。もっとも基本となる部分ですので，きっちりと把握しておいてください。

前回は，キーボードから入力された「式」を読み取ると同時に実行していました。キーボードから，

　Expression> 1+2+3

と打てば，即座に，

　Answer> 6

と表示されたはずです。

とりあえず，これで「式」の解釈はできるようになったのですが，この演算結果を格納できないと「プログラム」にはなりません。今回はこのプログラムを以下のように拡張して，変数を扱えるようにします（画面1）。

●入力
　変数＝式
　例：a = 1+2＊3
●出力
　いままで使用されたすべての変数とその値
　例）var1 = 0.123
　　　ver2 = 0.556
●終了
　なにも入力せずにEnterキー

変数とはなにか

言語処理プログラムを作る人に「変数」を説明しても，すでにわかりきっているでしょうが，いま一度，プログラミングを始める前に「変数」を定義しておきましょう。

1）変数は，名前で区別される。１文字目はアルファベット。２文字目以降はアルファベットか数字（例：value，value12，は変数。12Value は変数でない）
2）変数は値を持つ。値はプログラム中で定義され，のちに参照される

処理系で，「変数」を扱うというのは，この１）と２）を関連づけることにほかなりません。

今回はリスト1のように名前（m_pName）と値（m_dValue）の２つのメンバ変数を持つクラス（CVarTable）を定義し，この両者を扱うことにしました。

いまの段階では，C++のクラスを使わずに，C言語にもある構造体を使ってもよいのですが，処理系が複雑になると，

　変数の型（整数，実数，文字列……）
　変数の種類（グローバル，ローカル……）
　配列変数の次元

などの情報も同時に扱わなければならなくなってくるので，簡単に拡張のできる（継承できる），クラスを用いることにしました。

変数を扱う関数について

先月までは，各要素（厳密にいうと形態素とか終端記号とか，まぁtokenですね）を扱う際には，直接，対応するサブルーチンを呼んで，処理とともにトークンポインタ（char ＊tokenPtr）を進めていました。

変数を使う場合は，サブルーチン側でトークンポインタを進めずに，呼び出し側で管理するようにします。いままでどおりサブルーチンで管理しても問題はないのですが，変数の認識部分はバックトラックが多発するため，このようにしたほうがスマートにいくようです。

```
Expression> a=1+2*3
a        =7.000000
Expression> b=a*2
a        =7.000000
b        =14.000000
Expression> c=a+b
a        =7.000000
b        =14.000000
c        =21.000000
Expression> d=3
a        =7.000000
b        =14.000000
c        =21.000000
d        =3.000000
Expression> e=a+b+c+d
a        =7.000000
b        =14.000000
c        =21.000000
d        =3.000000
e        =45.000000
Expression>
```
画面1

変数テーブルを扱う関数は，主に，

　変数名の文字列の開始アドレス
　変数名の長さ

を与えて操作します。ユーザープログラム中に現れる変数名は，常に'¥0'で終わるとは限らず（"var1 = 1+2＊3"では，"= 1+2＊3"と続いていますね），どこで終わるかを処理関数に指定しなければならないので，変数名の長さも同時に与えます。

ラベルの長さを求める関数ですが，今回は，

　１文字目は，アルファベット
　２文字目以降は，アルファベットか数字

と定義したので，

```
int
VariableNameLength (void)
{
    char            *p = tokenPtr;

    if (isalpha (*p)) {
                    p++;
                    while (isalnum (*p)) {
                            p++;
    }
    }
    return ( p - tokenPtr) ;
}
```

と数えられます（リスト3）。この戻り値を，呼び出し側で保存しておき，処理が終わったら，

　tokenPtr += iLength;

として，トークンポインタを進めます。

リスト1　interpret¥table.h

```
class CVarTable
{
public:
   CVarTable();
   ~CVarTable();
   void    Display(void);
   CVarTable* RegisteNewValue(char *name, int length);
   CVarTable* Search(char *name, int length);
public:
   CVarTable* m_pNext;
   char*      m_pName;
   double     m_dValue;
};
```

このようにして用いる関数として，以下の２つが今回の目玉です。

●新しい変数を変数テーブルに登録する
CVarTable∗ CVarTable::RegisteNewValue（char ∗name, int length）；
●変数を変数テーブルから検索する。見つからなければNULL を返す
CVarTable∗　CVarTable::Search（char ∗name, int length）；

例）

```
int　iLength = ValueNameLength（）；　　// ラベルの長さを求める
CVarTable∗ pResult = VarTable.Search（tokenPtr, iLength）；
//　ラベルを探す
if（pResult != NULL）　　{
    // ラベルが見つかった
tokenPtr += iLength;　　// トークンポインタを進める
}
```

リスト２　interpret¥table.cpp

```
//
// 変数のテーブルを管理する
//

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

#include  "table.h"
#include  "MyProt.h"


// Constructor
CVarTable::CVarTable()
{
   m_pNext = NULL;
   m_pName = NULL;
   m_dValue = 0.0;
}

// Deconstructor
CVarTable::~CVarTable()
{
   if(m_pNext)
       delete m_pNext;

   if(m_pName)
       free(m_pName);
}

void
CVarTable::Display(void)
{
   CVarTable* p = this;

   while(p) {
       if(p->m_pName != NULL) {
           printf("%s¥t=%f¥n", p->m_pName, p->m_dValue);
       }
       p = p->m_pNext;
   }
}

CVarTable*
CVarTable::RegisteNewValue(char *name, int length)
{
   CVarTable* pLast = this;
   while(pLast) {
       if(pLast->m_pNext == NULL) break;
       pLast = pLast->m_pNext;
   }

   pLast->m_pNext = new CVarTable;
   CVarTable* pNew = pLast->m_pNext;

   char*  lpszName = (char *)malloc(length + 1);
   strncpy(lpszName, name, length);
   lpszName[length] = '\0';

   pNew->m_pName = lpszName;

   return pNew;
}

CVarTable*
CVarTable::Search(char *name, int length)
{
   CVarTable* p = this;
   while(p) {
       if(p->m_pName != NULL
       && !strncmp(p->m_pName, name, length)) {
           // Found!!
           return(p);
       }
       p = p->m_pNext;
   }
   // Not found
   return NULL;
}
```

変数の切り出し

以下のような入力を受け付ける際，最初に処理するのは，変数名の認識です。

Expression > Variable1 = 1+2∗3

リスト３　interpret¥main.cpp

```
// inter.cpp : コンソール アプリケーション用のエントリ ポイントの定義
//


#include  "stdio.h"
#include  "ctype.h"

#include  "table.h"
#include  "MyProt.h"

char  *tokenPtr;     /* トークン分析のためのポインタ */
char  inputBuff[300];

CVarTable  VarTable;


void
main(void)
{
   while(1) {
       printf("Expression> ");
       gets(inputBuff);
       tokenPtr = inputBuff;
       // 何も入力されなければ、終了
       if(tokenPtr[0] == '\0') break;

       CVarTable* pDestination = GetDestination();
       Check("=");    // 変数 "=" 式

       pDestination->m_dValue = ProcessExpression();
       if(tokenPtr[0]) {
           // <式>の後ろに、文字がある
           fprintf(stderr, "文法エラー\n");
       }
       VarTable.Display();
   }
}

CVarTable*
GetDestination(void)
{
   CVarTable *pResult;
   int iLength = VariableNameLength();

   pResult = VarTable.Search(tokenPtr, iLength);
   if(pResult == NULL) {
       // 見つからなければ新規に登録する
       pResult = VarTable.RegisteNewValue(tokenPtr, iLength);
   }
   tokenPtr += iLength;
   return(pResult);
}

int
VariableNameLength(void)
{
   SkipSpace();

   char   *p = tokenPtr;

   if(isalpha(*p))p++;
   while(isalnum(*p)) {
       p++;
   }
   return( p - tokenPtr);
}
```

キー入力の直後，行の先頭をTokenPtrは指していますから，以下のように順次処理します。

1）"Variable1"という変数を探す。なければ新規に登録する
2）"="を確認
3）式を処理（連載第1回を参照）
4）1）で探した変数に，3）の結果を代入

以上の処理は，リスト3の関数main（）中，以下のように処理されます。

```
CVarTable* pDestination = GetDestination（）;
Check（"="）;                       // 変数 "=" 式
pDestination->m_dValue = ProcessExpression（）;
```

実際には，これだけでは正しく動作が行われているかわかりませんので，代入文を実行するごとに，これまでに使われたすべての変数をその値とともに表示するようになっています（画面1）。

では，順に詳しく見ていきましょう。

関数GetDestination()は，まず，代入先の変数を変数テーブルから探します(CVarTable::Search(char *name, int length))。

ここで，変数テーブルを探しても使用された変数がない場合の処理は，処理系によって，

a）"未宣言の変数です"エラーとする
b）新たに登録する

の2種類考えられます。a)の代表例は，C言語ですが，iThisIsVariableとiThisIsValiableのようにタイプミスを発見するのに役立ちます。ただし，使用する変数すべてを事前に宣言しなければならないので，少しまどろっこしいのも事実です。b)の代表例はBASICですね。iThisIsVariableとiThisIsValiableのタイプミスは検出できませんが，変数が必要になったとき，宣言なしにすぐ使用できます。

大まかにいって，a)が，構造化を目指した大規模プログラミングを念頭に置いた言語処理系で採用されているのに対し，b)は，初心者向けやシェル言語など，手軽な処理系で採用されることが多いようです。

SX-BASICは，大規模プログラミングができるようa）を採用しましたが，とりあえず，今回は，まだまだ小規模な処理系ですので，b)を採用しましょう。具体的には，リスト3のGetDestination（）関数中，

```
pResult = VarTable.Search（tokenPtr, iLength）;
if（pResult == NULL）{
    // 見つからなければ新規に登録する
    pResult = VarTable.RegisteNewValue（tokenPtr,
iLength）;
}
```

です。もし，a)を採用する場合は，

```
pResult = VarTable.Search（tokenPtr, iLength）;
if（pResult == NULL）{
    fprintf（stderr, "未登録の変数です"）;
}
```

としてください。ただし，このままでは，ユーザープログラムから変数を登録する手段がありませんので，別途，追加する必要があります。

```
例）Expression> float var
    Expression> var = 1+2+3
```

これは，前回説明したAmatch（）関数("float"を検出できますね)とRegisterNewValue関数を使えば，簡単に実装できますので，各自研究課題としておきましょう。

変数の参照

以上で，変数に値を代入できるようになったわけですが，せっかく代入した値も読み出せなければ，価値が半減してしまいます。

```
例）  Expression> a = 1+2
      a    = 3
      Expression> b = a*2
      a    = 3
      b    = 6
```

さて，前回の復習になりますが，「式」の解釈は，演算子の優先度に応じて階層的に処理していました。「変数」は，「数値」の代わりに使われますから，「数値」と同じ，最下層の関数ProcessNumerics(void)で一緒に処理してしまいましょう（リスト4参照）。この関数中

a）isdigit(*tokenPtr)がTRUEなら「数値」として処理
b）isalpha(*tokenPtr)がTRUEなら「変数」として処理

と切り分けます。ただし，b)がいくら成り立っても，事前に登録されていなければ「変数」ではなく，ただの英数字文字列ですので，エラーを返します。

リスト構造

このように変数テーブルを用いれば，自作の処理系で「変数」を扱えるようになるのですが，その変数テーブルは，どのように管理すればいいのでしょう。

テーブルを管理するのに，いちばん簡単なのは，クラスCVarTableを配列として宣言することです。たとえば，

```
CVarTable   VarTable[100];
```

と宣言しておけば，

```
CVarTable*
Search（char *name, int length）
{
  int i;
  for（i = 0; i < 100; i ++）{
      if（!strncmp（VarTable[i].m_pName, name, length）
          return（&VarTable[i]）;;
  }
  return NULL;        // 見つからなかった
}
```

というように，簡単なループで扱えます。この方法は，扱いが簡単な半面，あらかじめ要素の数（この場合は100）を決めておかなければならないという制限があります。

言語処理系を設計する際，処理系の都合はなるべくユーザーに強要すべきではありませんから，今回は多少扱いが不便でも，リスト構造と呼ばれる可変個の要素を扱える方法を採用しました。

この方法は，

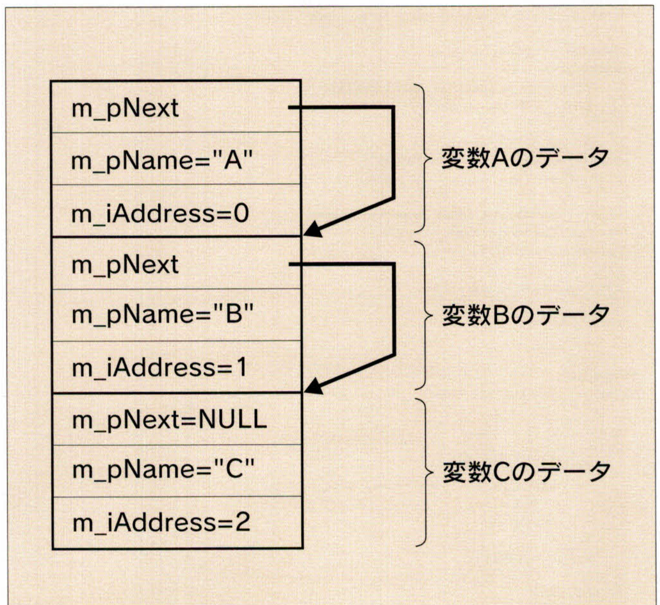


図1 リスト構造（変数A，B，Cがある場合のメモリダンプ）

リスト4　interpret¥exp.cpp

```
//
// 四則演算と括弧の処理を行うプログラム
//

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#include  "table.h"
#include  "MyProt.h"

extern char    *tokenPtr;
extern CVarTable  VarTable;

/*
** <式>の評価
*/
double
ProcessExpression(void)
{
   double  value = ProcessTerm(); /* <項>の収得 */

   while(1) {
       if(Amatch("+")) {
           value = value + ProcessTerm();
       } else if(Amatch("-")) {
           value = value - ProcessTerm();
       } else {
           break;
       }
   }
   return(value);
}

/*
** <項>の評価
*/
double
ProcessTerm(void)
{
   double  value = ProcessFactor();   /* <項>の収得 */

   while(1) {
       if(Amatch("*")) {
           value = value * ProcessFactor();
       } else if(Amatch("/")) {
           value = value / ProcessFactor();
       } else {
           break;
       }
   }
   return(value);
}

/*
** <要素>の評価
*/
double
ProcessFactor(void)
{
   double  value;

   if(Amatch("(")) {
       value = ProcessExpression();
       Check(")");
   } else {
       value = ProcessNumerics();
   }
   return(value);
}

/*
** <数値>の評価
*/
double
ProcessNumerics(void)
{
   double  value;

   SkipSpace();

   if(isdigit(*tokenPtr)) {
       /* 数字 */
       value = atof(tokenPtr);
       /* 読み終わった数字と小数点を飛ばす */
       while(isdigit(*tokenPtr) || *tokenPtr == '.') tokenPtr++;
       return value;
   } else if(isalpha(*tokenPtr)) {
       /* アルファベット。変数か？ */
       int length = VariableNameLength();
       CVarTable *pValue = VarTable.Search(tokenPtr, length);
       if(pValue) {
           /* 変数だった */
           tokenPtr += length;
           return pValue->m_dValue;
       } else {
           fprintf(stderr, "未宣言の変数です\n");
           return 0.0;
       }
   }
   /* 数字でも変数でもない */
   fprintf(stderr, "文法エラー\n");
   return( 0.0 );
}

/*
** 与えられた文字列を切り取れれば、文字列文tokenPtrを進め、 1を返す
** 切り取れなければ、tokenPtrを進めずに、0を返す
*/
bool
Amatch(char *s)
{
   char    *saveToken = tokenPtr;

   SkipSpace();

   while(*s++ == *tokenPtr++) {
       if(*s == '\0') {
           /* 比較する文字列の最後まで達した */
           return(1);
       }
   }
   /* 文字列は一致しなかった */
   tokenPtr = saveToken;
   return(0);
}

/*
** 与えられた文字列を切り取る。
** 取り出せなければ、文法エラー
*/
void
Check(char *s)
{
   if(! Amatch(s))
       fprintf(stderr, "文法エラー\n");
}


/*
** 空白文字列を飛ばす
*/
void
SkipSpace(void)
{
   while(isspace(*tokenPtr)) tokenPtr++;
}
```

a）各要素に，次要素へのポインタを持たせる（m_pNext）
b）最終要素はm_pNext == NULL;
c）新しく要素を足すときは，最終要素のm_pNextに継木する
ことにより実装されます（図1）。

先ほどの検索ルーチンだと，

```
CVarTable*
Search (char *name, int length)
{
  CVarTable *p;
  for (p = pStartTree; p; p = p->m_pNext) {
     if (!strncmp (VarTable[i].m_pName, name, length)
        return (&VarTable[i]) ;;
  }
  return NULL;        // 見つからなかった
  }
```

と検索できます。

ご察しのように，この方法には，

各要素に次要素のためのポインタが必要

n番目の要素を参照する場合，次要素をn回たどらなければならない

という欠点があります。また，DOSのように標準ライブラリのみ用いてプ

ログラミングしている分にはいいのですが，Windowsのようにメモリの確保・廃棄がしっかりと義務づけられている環境では，

メモリの破棄をすべての要素に対して忘れずに行わなければならないという手間もあります。これは，DOSだと，

変数Aのメモリ	←a
変数Bのメモリ	
変数Cのメモリ	
変数Dのメモリ	←b

と連続してメモリが確保されるので，a～bのメモリ領域をまとめて破棄すればよいのに対し，Windowsのようにマルチタスク環境では，ユーザーが途中でワードやエクセルを使わないとも限りませんから，

変数Aのメモリ	←a
ワードのなんかのメモリ	
変数Bのメモリ	
エクセルのなんかのメモリ	
変数Cのメモリ	
変数Dのメモリ	←b

のようにメモリ確保が途切れ途切れに行われる可能性があります。そのため，処理終了後，メモリ領域の破棄を行う際には，a～bの領域を一括して行うわけにはいかず，「変数Aのメモリ領域」「変数Bのメモリ領域」……とちまちま行う必要があります。

それでも，最小限のメモリを確保することにより，可変個の複数要素を操作できるというメリットに注目し，リスト構造を採用します。

制御構文

ここまでで，「式」および「変数」を扱える処理系ができました。これに，「制御構文」(goto文，if 文など)を加えれば，ひととおりの処理系を実現することが可能です。

ただし，あとで述べる理由により，これまでの方法に，制御構文を実装しても，あまり拡張性がないので，ここでは実例は示さずに，アイデアだけを紹介します(実装は，次回，中間コード形式で示します)。

「制御構文」とは，プログラムの流れを変えるための命令です。

リスト1～3のプログラムは，キーボードから1行入力されたら，入力された文字列の左端にtokenPtrをセットし，そこから解釈&実行する，というものでした。つまり，tokenPtrをセットしさえすれば，プログラムの途中でも，好きなところから実行できるということです。あとは，プログラムとtokenPtrにセットする値を対応づけするかという問題だけです。

● goto文

初期のBASICには，行番号という概念があり，ユーザープログラムすべての行に固有の番号がついていました。たとえば，

```
10 print "Hello World "
20 goto 10
```

というプログラムは，永遠に"Hello World "という文字列を画面に表示します。これは，"goto 10"を実行するときに，10行目の先頭にtokenPtrをセットすれば，実現できます。

もう少し高級なBASICになると，「ラベル」という概念を持っていました。

```
10 *Label1
20 print "Hello World "
30 goto Label
```

プログラマから見ると，格段に使いやすくなっているのですが，処理系自身から見るとやっていることにそう大差はありません。"goto Label"を実行するときに，プログラムの先頭から"*Label"という文字列を検索し，見つかったらその位置にtokenPtrをあわせればよいだけです。

●if文

```
if <式> then <文1> else <文2>
```

これも<式>の処理とgoto文の実現法がわかってしまえば簡単です。まず，<式>を評価します。これは，問題ありませんね(第1回参照)。

この値が0であったら，"else"というキーワード(予約語)を探して，その位置にtokenPtrをセットします。たいていは，"else"以降はなくてもよい

COLUMN 往年のBASICは，いかに行番号を扱っていたか？

中間コード形式でない8ビット時代のBASICは，いったいどのように行番号を扱っていたのでしょうか？　インタプリタを普通に実装すると，

```
10 print "Hello World !!"
20 goto 10
```

の"goto 10"というところで，プログラムの先頭から行番号10の位置を探して，そこにトークンポインタをセットします。

ですから，

```
10 ' This is comment
20 ' This is also comment.
30 print "Hello World !!"
40 goto 30
```

のようなプログラムでは，goto文の実行は，プログラムの先頭から，行番号10をスキップし，行番号20をスキップし，やっと行番号30にたどり着くわけです。そうすると，どちらのプログラムも，直前の行に制御を移すだけなのに，なぜか実行時間に差が出てしまう，という結果をもたらします。

この結果，プログラム高速化のテクニックとして，「よく使うサブルーチンは，前半に配置しろ」ということがいわれた時期もありました。

ところが，このような現象は，なぜかマイクロソフト系のBASICでは，起こりませんでした。以下に，そのからくりを紹介します。

当時は，拡張用16KB(＝0.016MB) DRAMが1万円していた時代ですから，BAISCのユーザープログラムもテキストそのままといわずに，予約語には固有の数値を使って圧縮がかけられていました。

例）"goto"　→　0x89
　　"print"　→　0x91

さらに，進んだ処理系では，自然数もそのままASCII文字列として扱わずに，16ビットの2バイト数とし，圧縮をかけていました。

例）"120"　→　0x0e 0x78 0x00
　(0x0eは，以下2バイト自然数が続く，というメタ記号)

以上のルールにより，

```
goto 120
```

と，8バイトの文字列が入力されたら，

```
0x89 0x0e 0x78 0x00
```

と4バイトに圧縮されてメモリに格納されます。

ところが，0x0dという数値を扱うもうひとつのメタ記号があり，これは，以下2バイトのアドレス情報が続く，ということを表します。

インタプリタが，

```
0x89 0x0e 0x78 0x00
```

を読み込むと，「えー，行番号10でもない，行番号20でもない，……行番号110でもない，アドレス0x8233の行番号120だー」ということになり，トークンポインタをアドレス0s8233にセットする前に先ほどのコード列を，

```
0x89 0x0d 0x33 0x82
```

というように書き換えます。2度目以降に実行するときは，飛び先はアドレス0x8233とわかっていますから，ユーザープログラムを先頭から探す手間をかけずに，トークンポインタを0x8233にセットできます。

このようにして，2回目以降の行番号→ユーザープログラムの格納先，の対応を調べる手間がはぶけるので，プログラムの前半のループと後半のループとで同じ実行速度が保てたのでした。

ただし，少しでもユーザープログラムに変更が加えられるとこの対応関係が崩れてしまうので，プログラムの実行が中断されたら，即座に，

0x0d　+　アドレス　→　0x0e　+　行番号

と，先ほどとは逆の方向に変換を行います。

今日では，テキスト自身を予約語表を使って圧縮するという必要があまりないので，以上のようなテクニックはあまり見なくなってしまいましたが，まぁ，昔はこのようなテクニックを使って，遅いCPUを酷使していたのだなぁという参考までに。

※　以上，予約語はNECのPC-8001の場合です。他機種では違う値が使われているかもしれませんが，考え方は，似ているはずです。

というオプションになっていますから，見つからなかった場合は，さっさとtokenPtrを次の行に進めます。

逆に<式>の値が0以外であったら，次のキーワードが"then"であることを確認し，<文1>を実行します。実行後は，そのままtokenPtrを進めずに，キーワード"else"があった場合には，それ以降をとばし，tokenPtrを次の行にセットし，<文2>が実行されないようにします。

●repeat～until <式>，while<式>～wend

さらに気の利いた処理系だと，goto文を使わなくてもプログラミングできるようにrepeat～until <式>，while<式>～wendのような制御構文が使えるようになっています。たしかに，構造化を意識したプログラムを作成するには大変便利ですが，処理するに当たっては，

repeat print "Hello World " until <式>	*Label1 print "Hello World " if <式> then goto Label1
while <式> print "Hello World " wend	*Label1 if <式> then goto Label2 print "Hello World " goto Label1 *Label2

と等価に処理してやればOKです。

少し深く考えると，ループが入れ子構造になっていたらどうするんだ，という問題もあります。その場合は，

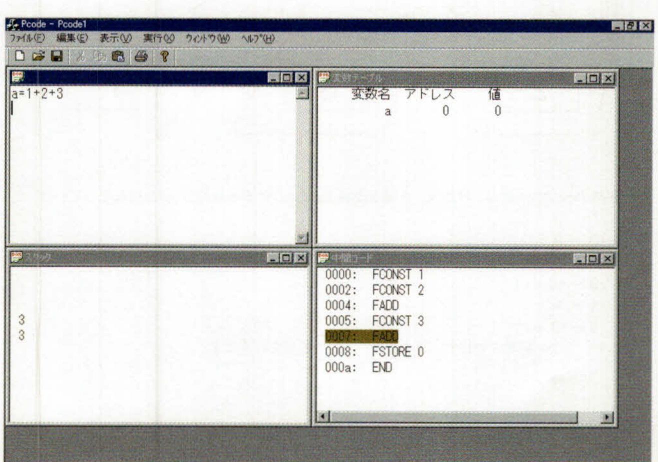
a) Label1，Label2の管理をスタック構造にする
b) ループ内の命令は再帰的に処理する

とだけいって各自の自由課題としましょう。純粋なインタプリタではありませんが，CD-ROM内のsxbasic¥sxbasic¥comp.cppのdoWhile ()，doRepeat () 関数内にb)を用いた一例が載っています。

と，以上で変数の管理を説明しました。実用的な処理系では，変数にローカル変数/グローバル変数の区別があったり，関数の管理も行わなければならなかったりと，いろいろ大変ですが，基本は同じです。

また，扱う変数の数が1000を超えるあたりから（変数というより，アセンブラのラベル管理など），変数の扱い方にも気をつけたほうがよい場合があります。これは，変数の検索時間が，変数の数に応じて指数関数的に増加するためです。これを防ぐには，「ソート」とか「検索」とか呼ばれるテクニックが有効なのですが，わりと体系化された教科書類があるので，必要な場合は各自で参照してください。

いままで作成してきたような，入力を即座に読み取り，実行する方式を



画面2

インタプリタ（Interpreter：通訳者）と呼びます。この方式は，構成が単純なため，電卓のように小規模な処理系を実現するには最適なのですが，解釈と実行が同時に行われるため，実行する処理量が多くなってくると，不利です。

それに対し，解釈と実行を分けたのがコンパイラ（Compiler：翻訳者）と呼ばれる処理方式です。

たとえば，100回繰り返しが必要な処理があったとしましょう。

インタプリタ：（解釈に1秒＋実行に1秒）×100 = 200秒

コンパイラ：解釈に1秒＋実行に1秒×100 = 101秒

というように，処理量が多くなると，解釈と実行を分けたほうがいい場合があります。

普通，C言語のコンパイラというと，

解釈部：プログラムを機械語に翻訳
実行部：その機械語を実行

のように処理を行います。

この方法は，実行時間が最短ですむという利点がある一方，プログラムの妥当性を解釈部でしか検証できないという欠点があります。たとえば，C言語で，

```
b[i] = 12;
```

という処理でアクセスするメモリエリアは，iの値がわからないと特定できません。iの値が特定できるのはこのプログラムの実行時だけです。実行するのは機械語ですから，いわれたとおりなにも考えず，指定されたメモリをアクセスします。プログラムミスで，iに不正な値が入っていた場合，そのアクセスはコンピュータを止めてしまったり，大事なほかのデータを書き換えてしまったりします。以前は，このような動作は致命的だったのですが，Windows 95以降では，「不正な命令を実行しました」とか「不正なメモリアクセスを行いました」などと，システムが感知して被害を最小限にとどめるような仕掛けがあります（運がよければ，の話ですが）。

そこで，コンパイラで，機械語に変換するのではなく，仮想的なコードに変換し，実行時にはその中間コードを解釈するという方法があります。中間コードは機械語そのものではありませんから，解釈のとき機械語に近い定義にしておけば，解釈時間を少なくすることは可能です（ただし間はある程度必要ですが）。

先ほどの例では，

プログラム→中間コードの解釈に1秒
＋（中間コードの解釈に0.2秒＋実行に1秒）× 100 = 121秒

となります。

また，最終的な制御はその解釈部が持っていますから，メモリアクセスを行う前に，本当にそのメモリアクセスが妥当かをチェックすることが可能です。つまり，コンパイラを中間コード形式にすることにより，

●実行時間は，純粋なコンパイラよりも必要（101 → 121秒）だが，インタプリタよりも高速に実行可能（200→121秒）
●実行時にプログラムの妥当性を検証できる

という両者の利点を組み合わせることができます。

たとえば，ネットワーク上で流通することを念頭に置くJavaでは，プログラムに悪意がないともいい切れないので，実行時に動作の検証を行わなければなりません。それと同時に，高速動作を行わなければならないので，中間コード形式が使用されています。

また，細かい点ですが，

●プログラムと中間コード用に，別々にメモリバッファが必要

という欠点もあります。組み込み用途やPDAでは，考慮する必要があるかもしれませんが，入門用のPCでさえ，32MBや64MBを実装している現在では，特に問題にはならないでしょう[注1]。

注1）ひとりの人間の書けるプログラムは，まぁ，1万行くらいが限度です。大雑把にいって，1行あたり10バイトの中間コードが必要だったとしても，1万行のプログラムで，10万バイト（=100KB）です。

中間コードを決める

以上に述べたように中間コードを用いて，SX-BASIC相当の処理系を作

るわけですが，どのようなコード体系を用いるかを決める必要があります。書籍によっては，方式を中間コード，コード体系を仮想コードと区別する場合があるらしいですが，以下，簡単のため中間コードで統一します。

とりあえず，ターゲットをPC互換機に絞っていますので，x86の機械語に近い中間コードのほうが解釈にかかる処理が少なくてすみます[注2]。

ただし，SX-BASICでは，インタラクティブな操作性を実現したいので，実行時にエラーが発生した場合には，すぐに，ユーザープログラム中のどこに発生したのかを発見できるようにします。そのためには，SX-BASICの文法と中間コードの体系があまりかけ離れていると，対応がつけづらくなってしまいます。

また，x86系のアーキテクチャはレジスタ型といって(基本的には)なにを行うにもレジスタを経由して行う，という処理体系です。つまり，x86系のアーキテクチャにするには，ユーザープログラムの演算を，x86のどのレジスタで行うかという「レジスタ割りつけ」をコンパイラが行わなければなりません。

この「レジスタ割りつけ」処理は，それなりに興味深いトピックではあるのですが，とても奥が深く，コンパイラの処理量が激増してしまいます。先ほどの，100回の繰り返しが必要な処理の場合，

プログラム→中間コードの解釈に100秒
　＋(中間コードの解釈に0.2秒＋実行に1秒)×100 ＝ 220秒

となってしまっては，インタプリタ形式で実行するより遅くなってしまい(200→220秒)，中間コード形式にした意味がありません。

そこで，プログラム→中間コードの解釈にも時間がかからず，中間コードの解釈にも，負担がかからないコード体系を定義します。

注2）世の中には，x86の機械語とまったく同じコードを使ったものもあります。たとえば，Windows 95のデバイスドライバは＊.VxDという形式で作成されていますが，これは，Virtual Device Driverの略で，いちおう中間コードで実行される，という前提に立っています。中間コードといってもx86のプロテクトモードの機械コードそのものなので，プログラムの際には，市販のCコンパイラなどを使用することが可能です。

また，WindowsNTでは，Windows 95用に書かれたプログラムをすべて中間コードとして実行します。不正なアクセスをOSレベルで検出できますので，Windows95などに比べてより安定した動作を行えます。

スタック処理

最近は，Java以外にあまりスタック処理方式を採用した中間コードを見なくなりました。もっとも，最近では言語処理系自体があまり話題にならないので，私が気づかないだけかもしれませんが……。

しかし，私の知る限り，数千行のユーザープログラムを実行する規模では，もっともバランスのとれた中間コードの体系です。拙作SX-BASICもこの方法でした。

スタック(Stack)とは，なにかを一時的に保存しておくからくりです。別名，LIFO(Last In First Out)というように，最後に入れられたものが，最初に出されていくというアクセス方式です。

コンピュータとは全然関係ありませんが，深夜営業の牛丼屋のカウンターにある，紅生姜も，このスタック方式で管理されています。お客さんが表面のほうから取れば，新鮮な(？)紅生姜のはずです。ただし，ざくっと取って中ぐらいになれば，昨日補充されたものが交じっているかもしれません。取り出せば取り出すほど，古い中身が出てきます。ある程度なくなってくると，また店員さんが上のほうに新しい紅生姜を補充しますのでとりあえず，いちばん上にある紅生姜はつねに新鮮なはずです(というわけで，容器の底には開店当時の紅生姜が残っている，という噂もありましたが，定期的に全部を廃棄しているみたいですね)。

このようにスタック式のアクセスは，あとに入れたものほど，早く取り出されていきます。なぜ，これが言語処理系でバランスがとれているのかを説明するのは，難しいですが，経験則からこういうことになっています。若い事務職員ほど，寿退社しやすい，という話もありますが，科学的に証明するのは困難です(酒の席では簡単なのですが)。

言語処理系では，このスタック処理を式の評価に使用します。たとえば，加算を行う際には，加数と被加数の2つの引数が必要ですが，中間コードの加算処理では，

1）スタックから値を取り出す
2）スタックから値を取り出す

レジスタ割りつけ

COLUMN

本文中で，難しい，複雑だ，といっておいてなにも説明しないのも後味が悪いので，ヒントだけ以下に説明します。

a+bのように単純な計算式の場合，左から，

変数a → r0
変数b → r1

と自動的に行えるので，レジスタ割りつけはそんなに難しくありませんが，

a＊b＋c＊d

のように，一時的な実行結果(a＊b)を(c＊dの計算中に)保存する必要がある場合，割りつけは複雑になります。Pentiumの場合，レジスタが8つあるので，

変数a → eax
変数b → ebx
変数c → ecx
変数d → edx

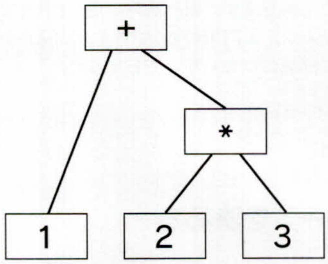
のようにいずれかのレジスタを保存用に使用できますが，

a＋b＊c＋d＊e＋f＊g＋h＊i

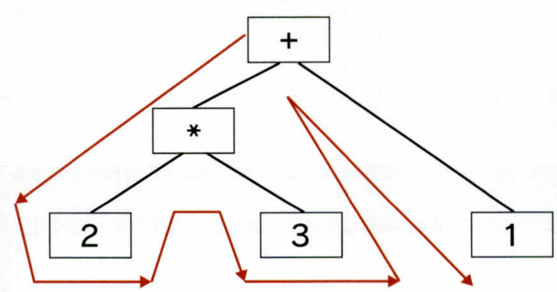
のように変数の数がレジスタの数を超えると，ちょっとした工夫が必要になってきます。

原則的に，式に使用される変数の数は無制限なので，このレジスタのやりくりを一般化しなければなりません。やり方は，式の解釈部でそのまま実行せずに，いったんツリー構造の解釈木を構成します。

例）1＋2＊3



ここで，階層の深い部分をなるべく先頭に持ってきます(a＋b＝b+aを思い出してください)。これは，階層が深い→変数が多い→一時記憶に使えるレジスタが少ない，ためです。



実際のレジスタ割りつけでは，上図を赤線のように「外側を反時計回り」になぞっていけば，

r0 ← 2
r1 ← 3
r0 ← r0＊r1
r1 ← 1
r0 ← r0＋r1

として，3つの要素に対して，2つのレジスタで処理を行えます。

リスト5　pcode ¥ exp.cpp

```
//
// 四則演算と括弧の処理を行うプログラム
//
#include "stdafx.h"

#if 0
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#endif

#include   "sxb.h"
#include   "table.h"
#include   "MyProt.h"

char   *tokenPtr;      /* トークン分析のためのポインタ */
extern ERRORCODE   i_ErrorCode;

// 変数テーブル
extern CVarTable   VarTable;

/*
** <式>の評価
*/
void
ProcessExpression(void)
{
   ProcessTerm(); /* <項>の収得 */

   while(1) {
       if(Amatch("+")) {
           ProcessTerm();
           SVM_PutCode(SVM_FADD);
       } else if(Amatch("-")) {
           ProcessTerm();
           SVM_PutCode(SVM_FSUB);
       } else {
           break;
       }
   }
}

/*
** <項>の評価
*/
void
ProcessTerm(void)
{
   ProcessFactor();    /* <項>の収得 */

   while(1) {
       if(Amatch("*")) {
           ProcessFactor();
           SVM_PutCode(SVM_FMUL);
       } else if(Amatch("/")) {
           ProcessFactor();
           SVM_PutCode(SVM_FDIV);
       } else {
           break;
       }
   }
}

/*
** <要素>の評価
*/
void
ProcessFactor(void)
{
   if(Amatch("(")) {
       ProcessExpression();
       Check(")");
   } else {
       ProcessNumerics();
   }
}

/*
** <数値>の評価
*/
void
ProcessNumerics(void)
{
   /* 空白文字列を飛ばす */
   SkipSpace();

   if(isdigit(*tokenPtr)) {
       SVM_PutCode(SVM_FCONST);
       SVM_PutCode(atoi(tokenPtr));
       /* 読み終わった数字を飛ばす */
       while(isdigit(*tokenPtr)) tokenPtr++;
   } else if(IsLabel1stChar(*tokenPtr)) {
       int length = VariableNameLength();
       CVarTable   *label = VarTable.Search(tokenPtr, length);
       tokenPtr += length;
       if(label) {
           SVM_PutCode(SVM_FLOAD);
           SVM_PutCode(label->m_iAddress);
       }
   } else {
       /* 数字でも変数でもない */
       i_ErrorCode = ERROR_SYNTAX;
   }
}

/*
** 与えられた文字列を切り取れれば、文字列文tokenPtrを進め、 1を返す
** 切り取れなければ、tokenPtrを進めずに、0を返す
*/
int
Amatch(char *s)
{
   char    *saveToken = tokenPtr;

   /* 空白文字列を飛ばす */
   while(isspace(*tokenPtr)) tokenPtr++;

   while(*s++ == *tokenPtr++) {
       if(*s == '\0') {
           /* 比較する文字列の最後まで達した */
           return(1);
       }
   }
   /* 文字列は一致しなかった */
   tokenPtr = saveToken;
   return(0);
}

/*
** Amatch + 与えられた文字列の後ろが空白記号であること
*/
BOOL
AmatchS(char *s)
{
   char    *saveToken = tokenPtr;

   if(Amatch(s)
   && isspace(*tokenPtr))
       return(TRUE);

   /* 文字列は一致しなかった */
   tokenPtr = saveToken;
   return(FALSE);
}

/*
**
*/
void
Check(char *s)
{
   if(Amatch(s) == FALSE) {
       i_ErrorCode = ERROR_SYNTAX;
   }
}

/*
** 空白を読み飛ばす
*/
void
SkipSpace(void)
{
   while(*tokenPtr && isspace(*tokenPtr))
       tokenPtr++;
}
```

3）1）と2）の値を加算する
4）演算結果をスタックに積む

という動作を行います。このようなスタック動作では，加数をレジスタr0に割り当てて，非加数をr1に割り当てて……などと，いうレジスタ割りつけを考えなくてすむので，効率的です。

ここで，スタック処理を基本とした中間コードのコード体系を決めるのですが，今回は表1のようにしました。前回までに行っている処理を，ざっと見て，見つけた順に1から値をつけていったもので，値自身に深い意味はありません。このコードを用いると，たとえば，

a = 1 + 2 * 3

というユーザープログラムは,

```
fconst  1     // 定数1をスタックに積む
fconst  2     // 定数2をスタックに積む
fconst  3     // 定数3をスタックに積む
fmul // スタックのいちばん上の値(3)と2番目の値(2)を乗算し
(3×2→6), その結果をスタックに積む
fadd // スタックのいちばん上の値(6)と2番目の値(1)を加算し
(6+1→7), その結果をスタックに積む
fstore0//  スタックのいちばん上の値(7)を変数0(a)に代入する
   end
//  終了
```

となります(スタックの状態は図2)。

中間コードの実装

注意深く観察すると，これまでのプログラムは，ユーザープログラムの解釈と同時に実行をやっていました。たとえば，"*"を解釈する部分(exp.cpp中のProcessTerm関数)では，

```
if (Amatch ("*")) {
    value = value * ProcessFactor () ;
}
```

と，"*"を見つけると同時に乗算を行っていました。解釈と実行を分ける中間コードを用いるわけですから，

```
if (Amatch ("*")) {
    ProcessFactor () ;
    乗算をするための中間コードを出力
}
```

と変更します(リスト5)。もし，Windows9x環境をお持ちでしたら，CD-ROMのsxb/pcodeに図2のようなプログラムが入っていますので，起動し，プログラム起動時に表示されるウィンドウに，

a=1+2+3

などと，「変数」「=」「式」の順に入力し，

実行 | Compile (あるいはF7)

中間コードを作成

ウィンドウ | 中間コード

で，各プログラムがどのように中間コードに変換されるのかを見てください。また，

ウィンドウ | 変数テーブル

で，使用された変数の値が表示されます。

実行 | Step

で，中間コードを1つひとつ実行しますので，

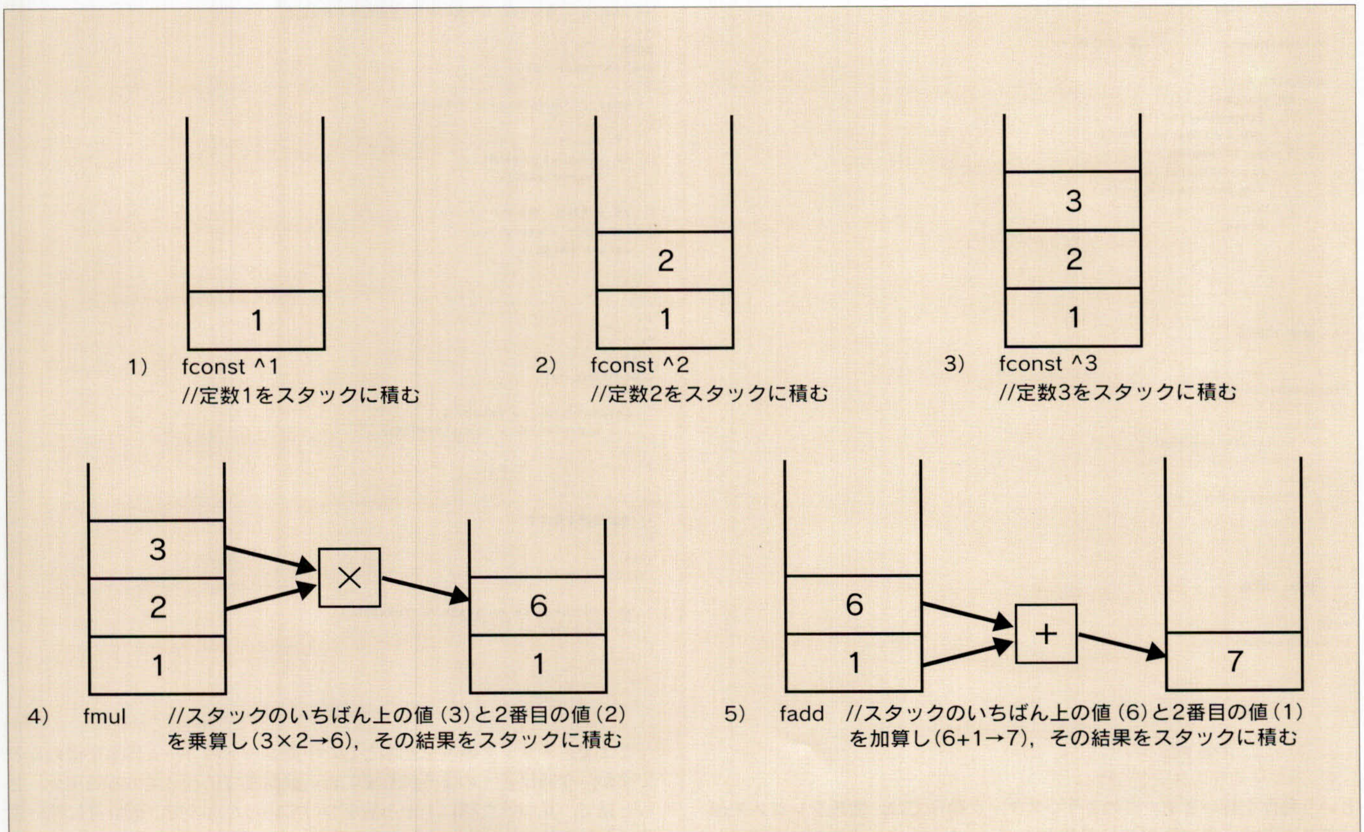
ウィンドウ | スタック

で，スタックの状態を調べてください。式の計算が終了し，変数に値が代入されると，表示されている変数の「値」の欄が変化するはずです。

いろいろな式を試すとやがて，「式に現れる定数や変数の順序が，ユーザープログラムと中間コードで同じ」ということがうすうすわかるのではないかと思いますが，これはスタック型中間コードの特徴で，バックトラックを抑えられ，処理系が簡単になるというメリットです。

Windows版

Oh!Xのレイアウトには，いろいろ賛否両論あるみたいですが，前回の記事は，書いた自分でも驚いているのですが，よいですね。ま，記事の内容はたんなる技術の解説ですが，全ページカラーで，いやぁ，状態遷移の図なんか，すばらしい。こういう専門的な話はたいてい白黒と相場が決まっていたのですが，カラーにしてみると説得力の次元が違いませんか？



画面2 スタック処理(a＝1＋2＊3の例)

表1　中間コード

コード	ニーモニック	内容
0 n	SVM_FCONST	nをスタックに積む
1 n	SVM_FLOAD	アドレスn(で表される変数)の値をスタックに積む
2 n	SVM_FSTORE	スタック上の値をアドレスn(で表される変数)に代入する
3	SVM_FADD	スタックのいちばん上の値と2番目の値を加算し，その結果をスタックに積む
4	SVM_FSUB	スタックのいちばん上の値と2番目の値を減算し，その結果をスタックに積む
5	SVM_FMUL	スタックのいちばん上の値と2番目の値を乗算し，その結果をスタックに積む
6	SVM_FDIV	スタックのいちばん上の値と2番目の値を除算し，その結果をスタックに積む
7	SVM_FREM	スタックのいちばん上の値と2番目の値を余算し，その結果をスタックに積む
−1	SVM_END	中間コードに実行を終了

表2　中間コード形式の処理系に使われているファイル

ファイル	処理内容
式，変数を処理するためのファイル	
exp.h exp.cpp	「式」の評価
state.h state.cpp	「文」の評価
svm.h svm.cpp	中間コードの実行
table.h table.cpp	変数テーブルの管理
Windowsのためのファイル(MFC)	
ChildFrm.h ChildFrm.cpp	MDIの子ウィンドウフレームをサポート
MarinFrm.h MainFrm.cpp	MDIの親ウィンドウフレームをサポート
Pcode.h Pcode.cpp	プログラムの初期化/終了処理
PrgView.h PrgView.cpp	中間コードを表示(表示のみ，編集などはしない)
SrcDoc.h SrcDoc.cpp	ユーザープログラムの表示/編集などをサポート
StdAfx.h StdAfx.cpp	コンパイル時間を節約するためのファイル
StkView.h StkView.cpp	スタックの状態を表示（表示のみ，操作はしない）
TableView.h TableView.cpp	変数テーブルを表示（表示のみ，操作はしない）

少し大きな書店に行けば，コンパイラの作成法の書籍は数十冊くらいあるのですが，たいていは難解で抽象的な専門書だったり，実際例の載っていない教科書だったりするのが不満で始めたこの連載ですが，カラーページをいただけるとなると，もう少し私もがんばらなければなりません。

今回は，プログラムをWindows 95対応にしてみました。

スタック操作とか中間コードなどは，一度理解してしまえば，どうということはないのですが，第一歩が踏み出せないとなかなか理解が難しいトピックスです。いくら，カラーページで図をふんだんに使っても，図は図です。動きません。スタック操作のように，各ステップごとに状態が変わる現象を説明するためには，コンピュータで実際に動くプログラムの内部状態をグラフィック表示すれば，格段にわかりやすくなるはずです。

Windows対応にしたので，処理系にとって本質的でない部分(ウィンドウを開く/閉じるなど)をサポートするため，多数のファイルがありますが，以上説明した動作を実現するのに必要なファイルは以下の4つのみです。

exp.cpp / exp.h　「式」の評価
state.cpp / state.h　「文」の評価

今回の記事の前半部分で説明した内容です。ただし，ユーザープログラム解釈時には実行を行わず，中間コードを出力します。

table.cpp / table.h　変数テーブルの管理

今回の記事の前半部分で説明した内容です。各変数情報は，値(m_dValue)だけでなく，アドレス情報(m_iAddress)を持っています。

svm.cpp / svm.h　中間コードの実行

表1にある中間コードを解釈し，実行します。

残りの参考までに，残りのファイルとその役割を表2に挙げておきましたので，興味のある方はMFC入門とあわせてご覧ください。

付録CD-ROMの内容

今回，「変数の管理」と「中間コード」の2つのトピックを扱ったため，今回のCD-ROMには，いろいろなプログラムを収録させていただきました。

SXBASIC ¥ interpret　テーブル管理つきインタプリタ(DOS版)
SXBASIC ¥ pcode　中間コード形式の処理系(Windows版)
SXBASIC ¥ SXBASIC　SX-BASIC(未完成)

最初のテーブル管理つきインタプリタは，構造が簡単なDOS版のプログラムです。DOSに固有の命令は使っていないので，ほかのシステムでもコンパイルし直せば動作するはずです。

2つ目は，前述のように中間コード形式の処理系です。Windows 95依存部分をなんとかすれば，ほかの処理系に持っていくことは可能でしょう。

最後のSX-BASIC(未完成)というのは，SX-WINDOW用のSX-BASICをWindows95用に移植したものです。だいたいの命令・関数は処理が終わっていますが，ファイル関数，リッチテキスト(SX-WINDOWでいうマルチフォントテキスト)，リソース，クリップボード，ウィンドウエンジンなど，システム依存の部分が終わっていません。あと，

- IE 5.0をインストールしてあるWindows 98では，コンソールウィンドウを開いた状態で，ソースウィンドウを閉じて，アプリケーションを終了すると止まる
- Windows 95 Retail Version(OSR2でないもの)で，dskf()関数がまともに動かない

などといくつかの問題をはらんでいます(菊地君，今度とも末永くよろしく)。実用にはいま一歩といった感じなのですが，それでも，

４種類(char, int ,float, str)の変数の型
配列変数
関数定義・呼び出し
制御構文

など，本文では説明しきれなかったトピックが実装されていますので，必要に応じて参照してください。使用方法は，

File | Open　プログラムの読み出し
File | Save　プログラムの保存
Execute | Run　プログラムの実行

となっています。また，

Window | Console　コンソールウィンドウを開く
Window | Stack　スタックの状態を表示する
Window | Disassemble　中間コードを表示する

と，内部状態が調べられるようになっています。

仏日翻訳ソフト「来来仏語」 第3回

日本語の組み立て

石上 達也 Ishigami Tatsuya

独自の翻訳エンジンを持つ来来仏語。よりよい日本語に変換するためには文法解析などに多少の工夫が必要です。今回は主に日本語文法のシステム化に関した話題を取り上げてみましょう。

中学生の頃から「Oh!X」編集部におじゃましているので，私が文章の書き方を意識するようになって，十数年経ちます(そういえば，編集の(U)氏は，当時，新卒で出版事業部に配属されたばっかりだったなぁ……)。当時は，新製品レビューのほかは，ほとんどの記事が自作プログラムの発表＋解説という感じで，誌面の多くの部分をプログラムリスト（のプリンタ出力）が占めていました。

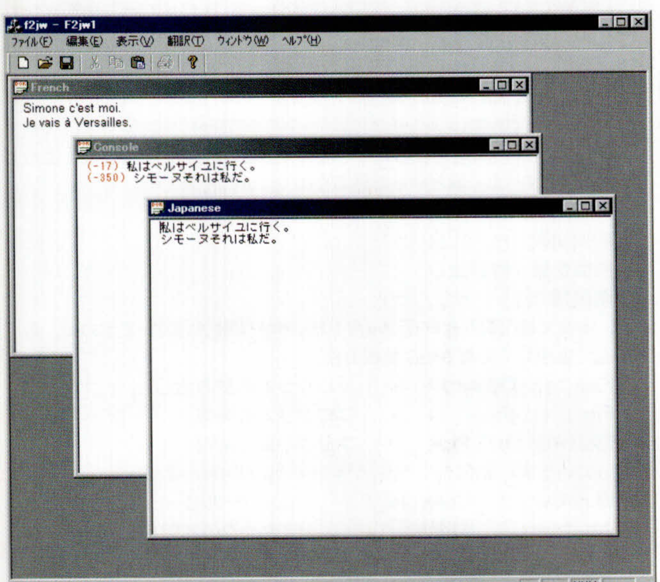
私も，プログラムを3～4カ月くらいかけて作成し，2～3日で文章を書くというように，プログラム作成を主体に活動していました。ところが，自分の作成したプログラムだけでなく，入門記事の執筆などの機会が増えるにつれ，必要な情報量を持った文章だけでなく，いかにわかりやすく書くか，ということも気を遣いはじめました。記事の構成だけでなく，同じ単語を使った文章でも，並べ方や句読点の位置などで，より伝わりやすくなります。それまで，ほとんど意識しなかった文章作法などの参考書を買ったりもしました。

「来来仏語」開発当初，市販の英日翻訳プログラムの出力結果をいろいろ調べてみたのですが，日本語の単語の順序を変えるだけでずっとわかりやすくなるのにとか，助詞を切り替えるだけで日本語の品質がずっと上がるのにとか，日本語の文章構築で手を抜いてしまったため，翻訳プログラム自体が貧弱に見えてしまうことがたびたびありました。

「来来仏語」は，まだまだ未完成で，構文すらとれない仏文がかなりありますが，日本語の出力には，わりと気を遣っています。「来来仏語」は大まかにいって，以下の3つの要素から成り立っているのですが，今回は，3)の部分を解説します。

1）入力された仏文をリスト構造へ変換
2）プログラムされた文法規則にあうようにリスト構造データを変換
3）構文解析されたリストデータを日本語として出力

改めて，コンピュータに日本語の構成を教え込んでみると，新しい発見が相当ありましたので，この記事が翻訳プログラムを作る際だけでなく，逆に文章を書く際にも参考になればと思います。



画面1 ついにWindows版となった来来仏語

「来来仏語」の構成

少し間があいてしまったので(って，10カ月も前ですね)，第2回目の内容を簡単に復習すると，

・入力された仏語は，リスト構造に変換される
・リスト構造の連結は，

Subject:	動作の主格
Child	動詞の副詞 or 名詞の形容詞
Object1	目的語1
Object2	目的語2

の4種類ある
・文章は，動詞Tokenに結合される

ということでした(図1)。

仏語から，どのようにリスト構造を組み立て，評価するかというのは，まだ作業中ですし，仏語にとても依存した話ですので，しばらく私の個人的な楽しみとし，別な機会に解説します。今回は，このリスト構造から，どのように日本語を生成するのか，がテーマです。

助詞

名詞だけで成り立っている文章などを除いて，文章(＝主語＋述語)が正しく翻訳されると，すべて単語は，動詞Tokenの子Tokenとして，連結されます。

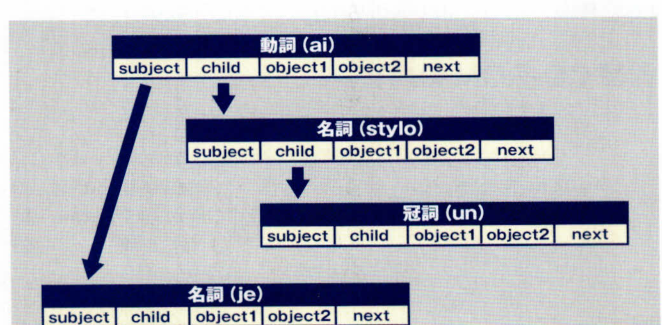


図1 リスト構造の例 J'ai un stylo.（I have a pen.）

その連結の結果によって，動詞のもっとも適切な日本語訳が選ばれます。別のいい方をすると，（名詞や形容詞ではなく）選ばれる動詞訳に，日本語訳はもっとも影響を受けます。

たとえば，一般的な日本語の構文を，

「主語」＋"は"＋「目的語」＋"を"＋「動詞」

と規定すると，

J'ai une voiture. (I have a car.: 私は車<を>持っている)

に当てはまりますが，

J'aime vin. (I like win.：私はワイン<を>好きだ)

では，不自然な感じがします。ここは，

「私はワイン<が>好きだ」

としたいものです。

このような，「を」→「が」の切り替えの一般ルールを見つけだしプログラミングできればよかったのですが，なかなか難しく，各動詞訳（クラス名：JP_VERB）に埋め込んでしまいました。

さらに，同じ動詞のai（原型はavoir，英語のhaveに相当）を使っても，

J'ai une voiture. (I have a car. ：私<は>車を<持っている>)

J'ai une soeur. (I have a sister. ：私<には>姉妹が<いる>)

という具合に，日本語訳によって，使用する助詞も変えたほうがよい，という問題も，各動詞訳に助詞を埋め込むことで解決できます。

各動詞訳に助詞を埋め込んだところで，たいしたデータ量ではないので，いまのところこの方法を用いていますが，もし，助詞の選択に関して，アドバイスをお持ちの方がいらっしゃいましたら，編集部あるいは，

http://www.softbank.co.jp/publishing/ohx/

のStudio On lineまで，ご連絡お待ちしております。

助詞を表すメタ記号

さて，動詞訳に助詞を埋め込むのはよいのですが，文章によっては，同じ動詞でも，副詞や構文次第で助詞を変えたほうがよい場合があります。

例） J'ai une voiture.
(I have a car.：私<は>車を持っている)
J'ai aussi une voiture.
(I also have a car.：私<も>車を持っている)

これは，副詞aussi（英語のtooとかalsoに相当）があるため，「は」→「も」と変換したのですが，変換する際，「来来仏語」にどれが助詞かを認識させる必要があります。そこで，助詞をそのままシフトJISコードで辞書中に埋め込むのではなく，一種のメタ記号として表し，出力する際に，助詞の入れ替えを検討します（実際の処理は，ファイル「print.cpp」中の関数PrintOutJpPrepositionで行っています）。

このメタ記号は，

- 1文字目は，"%"で始まる
- 2文字目は，表1のいずれか。3文字目に動詞・形容詞・形容動詞などを使用する際の，活用形を指定（オプション）
- 3文字目は，以下のいずれか
  - 0 （Subjectの内容を表示）
  - 1 （Object1の内容を表示）
  - 2 （Object2の内容を表示）

  （Subject，Object1，Object2に関しては，図1を参照）
- 4文字目は，表2のいずれか

表2 助詞を表すメタ記号

メタ記号	対応する助詞	置換される可能性のある助詞
A	のことが	のほう，のことしか，のことだけ，のことでも，のことも，のこと
C	に	にも，しか，だけ，には
D	で	だけ
E	へ	へしか，へも，へだけ
F	ことが	ことのほう，ことしか，ことだけ，ことも
G	が	のほう，しか，だけ，も
H	は	こそ，も，が，だけが
I	に	だけ，にも
J	ことに	ことだけに，ことにも
K	から	だけから
L	への	へだけの，へも
N	の	だけの
S	として	だけとして
T	と	のほうと，だけと
W	を	しか，だけ，でも，も
Y	よりも	

表1 活用の種類

活用形	メタ記号	ラベル	例				コメント
未然形	n	JP_CHANGE_NEGATIVE	運ば/ない	見/ない	高く/ない	静かで/ない	
		JP_CHANGE_MO	運びもし/ない	見もし/ない	高くも/ない	静かでも/ない	
	l	JP_CHANGE_LETS	運ぼ/う	見よ/う	高くなろ/う	静かにしよ/う	
	r	JP_CHANGE_RESPECT	運ば/れる	見ら/れる			
	f	JP_CHANGE_FORCE	運ば/せる	見さ/せる			
連用形	p	JP_CHANGE_PASSE	運ん/だ	見た	高かっ/た	静かだっ/た	「た/だ」と接続する場合の音便を考慮した連用形
	g	P_CHANGE_PLEASE	運んで/下さい	見て/下さい	高くして/下さい	静かにして/下さい	
	z	JP_CHANGE_CONT	運び	見て	高く	静かに	連続する場合の連用形 例）走り，飛ぶ
	c	JP_CHANGE_ADVERB	運び	見て	高く/なる	静かに/なる	
	h	JP_CHANGE_TAI	運び/たい	見/たい	高くし/たい	静かにし/たい	「たい」に接続
終止形	s	JP_CHANGE_STOP	運ぶ	見る	高い	静かだ	
		JP_CHANGE_QUESTION	運ぶ/か	見る/か	高い/か	静か/か	
連体形	a	JP_CHANGE_ADJECTIVE	運ぶ/とき	見る/とき	高い/とき	静かな/とき	
		JP_CHANGE_EXCLAME	運ぶの/だろう	見る/だろう	高いの/だろう	静かなの/だろう	感嘆文の訳に使用
已然形	i	JP_CHANGE_IF	運べ/ば	見れ/ば	高けれ/ば	静かなら/ば	
	e	JP_CHANGE_ABLE	運べ/る	見れ/る			
命令形	o	JP_CHANGE_ORDER	運べ	見ろ			
語尾なし		JP_CHANGE_NONE	運	見	高	静か	語尾なし

というルールで一般の文字列とは区別されます。

例） J'ai une voiture.

(I have a car.：私は車を持っている)

の翻訳には以下のテンプレートが選ばれます(ファイルdic_verb.cppに入っています)。

"%0H.%1W.持って", JP_KIND_IRU

これを，

・%0H → 「主語」(Subjectとして連結)＋"は"
・%1W → 「目的語１」(Object1として連結)＋"を"
・"持って"
・JP_KIND_IRU → "いる"(後述)

という変換を行い，日本語を出力します。

活用形の拡張

日本語の動詞は，否定(持っていない)，過去(持っていた)などと，語尾を活用する必要があるので，

"%0H.%1W.持っている"

とはせずに活用の種類(JP_KIND_IRU)を指定し，

"%0H.%1W.持って", JP_KIND_IRU

という形でテンプレートを作成し，仏文の構文解析が終わった時点で，肯定・否定や時制を吟味したうえで，適切な訳(動詞の語尾)を出力します。

さて，この日本語の活用形ですが，辞書の付録資料などに紹介されているのは，五段活用，上一段活用，下一段活用，形容詞型活用，形容動詞型活用，カ行変格活用，サ行変格活用と特殊な助動詞がいくらかでしょう。

これらの分類で日本語を扱うこともできるのですが，例外が多すぎてコンピュータで扱うには適しませんので，試行錯誤の末，表3のように拡張しました。

たとえば，ある動詞は，活用形(例：ア行五段活用)と活用(例：未然形)どおりに杓子定規に語尾をつければ，一応の日本語にはなるのですが，不自然に聞こえる場合があります。

例） 終止形：(机の上にペンが)ある
未然形：(机の上にペンが)あらない → ない

また，活用によってはツ音便というものが発生する場合としない場合があるのですが，これも２種類に分けます。

(普通の)カ行五段活用(JP_KIND_GODAN_KA)書<い>た

(音便付き)カ行五段活用(JP_KIND_GODAN2_KA2)行<つ>た

また，厳密にいうと活用ではありませんが，仏文に進行形的な要素がなくても，日本語にしたとき「〜っている」と記したほうが自然な動詞は，プログラム内部で別の活用形として扱っています。

例） Je le connais. (I know him. ただの現在形。進行形ではない)
私は彼を知る → 私は彼を知っている

ただし，このconnais(原型はconnaitre)の訳を，

"%0H.%1W.知って", JP_KIND_IRU

としても，肯定形では同じ結果が得られますが，否定形にしたとき不自然になってしまいます。

例） Je ne le connais pas. (I don't know him.)
私は彼を知っていない → 私は彼を知らない

この切り分けを自動的に行うため，この「知る」という動詞をラ行五段活用(JP_KIND_GODAN_RA)ではなく，第２ラ行五段活用(JP_KIND_GODAN2_RA)という別の活用として登録しています。

〜しなければならない Devoir(Must)

日本語には，「義務」「強い推定」を一語で表す助動詞はありませんから，「〜しなければならない」「〜に違いない」と，「動詞」＋「ない」を用いて同等の意味を表します。「〜する必要がある」「〜と強く思われる」といい回すこともできますが，とても不自然です。

両者は，ともに形容詞型活用を持つ助動詞「ない」を語尾に持っていますので，肯定形は，形容詞型の語尾テーブルで処理できます。

例） Je dois y aller.
(I have to go there.：私はそこへ行かなければならない)
J'ai dù y aller.
(I had to go there.：私はそこへ行かなければならなかった)

しかし，言い切りの形が，もともと否定形を表す助動詞ですから，

Je ne dois pas y aller.
(I don't have to go there.：
私はそこへ行かなければならなくない)
Je n'ai pas dù y aller.
(I don't have to go there.：
私はそこへ行かなければならなくなかった)

と，不自然です。

そこで，助動詞「ない」と，活用テーブルを共有するのをやめて，「〜しなければならない」を，表4のように特別な活用形を持つ助動詞として扱うことにしました。

「は」と「が」

「西洋人が最も困難を感ずるのは，主格を現すテニヲハの「ハ」と「ガ」の

表4 特殊な活用形

活用形	メタ記号	ラベル	例	
			JP_KIND_FALLOIR	JP_KIND_POUVOIR
未然形	n	JP_CHANGE_NEGATIVE	はいけ/ない	とは限ら
		JP_CHANGE_MO	はいけなくも/ない	とは限らなくも
	l	JP_CHANGE_LETS	必要としよ/う	
	r	JP_CHANGE_RESPECT	必要とさ/れる	
	f	JP_CHANGE_FORCE	必要とさ/せる	できるようにさ
連用形	p	JP_CHANGE_PASSE	必要があった	かもしれなかった
	g	JP_CHANGE_PLEASE	必要として	かもしれない
	z	JP_CHANGE_CONT	必要とし	かもしれず
	c	JP_CHANGE_ADVERB	必要とし	かもしれない
	h	JP_CHANGE_TAI	必要とし	かもしれなくし
終止形	s	JP_CHANGE_STOP	必要がある	かもしれない
		JP_CHANGE_QUESTION	必要がある	かもしれない
連体形	a	JP_CHANGE_ADJECTIVE	必要な	かもしれない
		JP_CHANGE_EXCLAME	必要とするの	かもしれないの
已然形	i	JP_CHANGE_IF	必要とすれ	かもしれなけれ
	e	JP_CHANGE_ABLE	必要とでき	かもしれない
命令形	o	JP_CHANGE_ORDER	必要としろ	かもしれない
語尾なし		JP_CHANGE_NONE		

表5 アクセント記号

仏語	成り立ち	キーボード入力文字
à	a + accent grave	@
â	a + accent corcpmf;exe	[
ç	c + c{dille	¥
è	e+accent aogi	{
é	e + accent grave	}
ê	e + accent circonflexe	]
ë	e + tr{ma	&
î	i + accent circonflexe	^
ï	i + tr{ma	%
ô	o + accent grave	`
ù	u + accent grave	|
û	u + accent circonflexe	‾

区別だそうでありますが，なるほど，「花は散る」と云うのと「花が散る」と云うのと，明らかに使い道が違っておりまして，われわれならその場に臨んで迷うことはありませんけれども，さてそれを，一般に当て篏まる規則として，抽象的に云えと云えば出来ない」　（谷崎　潤一郎「文章読本」）

とはいっても，これをなんとか一般化しなければ，翻訳ソフトは作れないわけで，以下は，一応，私なりに一所懸命考えた結果です。

ちなみに，現バージョンの「来来仏語」では，目的語を取らない自発的動作は，すべて「が」と処理しています。

例）　Les arbres fleurissent.（The trees flowered.）
（それらの）木が咲いていた

●無条件に主語＋「が」を表示する動詞

はっきりとはいいきれないのですが，通常ではあまり起こらないような動作，状態は「は」ではなく「が」を使ったほうがよいみたいです。

例）　La guerre éclata.　（The war occurred.）
（その）戦争<が>勃発した

「（その）戦争<は>勃発した」だと，「しかし，人々は平和に暮らした」とか「別の戦争は勃発しなかった」とか，次の文章に変な期待がかかってしまいます。主格を「が」にすることで，次の文章に依存しないで，独立した文章になります。

そのほかに，日本語訳する際に，受動態で訳したほうがすっきりくるものがあります。

例）　L'incendie se voit de ma fenêre.
（英語にないいい方だが，無理やりいうと The fire is visible from my window.）
（その）火災<が>私の窓から見える

●主格を尋ねる疑問文

例）　Qui a trouvé ce histoire?（Who found this history / story?）
誰<が>この歴史/話を見つけたか？

●重文，複文中の従文

ネストされた内側の文章は，主語に続く助詞を「は」→「が」へと変換します。「が」のほうが，主語の近くに述語を期待させるので，入れ子構造が明確になると思います。

例）　Il dit que j'ai une voiture.（He said that I have a car.）
私<が>車を持っていると，彼はいった

●関係代名詞の従文

例）　Une voiture que il a acheté.（A car which he bought.）
彼<が>買った車

理由はよくわからないのですが，「彼は買った車」というと不自然ですね。

●代名詞「誰か」「全員」

この２つの代名詞は，無条件に「が」を従えるようにします。

例）　Quelqu'un a frappé à porte.　（Someone knocked the door.）
誰か<が>ドアをノックした

例）　Toute le monde chante.　（Everyone sings.）
全員<が>歌う

ちなみに，参考文献１によると，「は」と「が」の違いを考えるとき，
文章＝主語＋「は/が」＋述語
という西洋言語の型にはめて考えるより，「は」は以下の４つのいずれかとなる。

１）話の場を設定する/題目を提示する

例1）　MSのスケジュール<は>，いつも遅れる
例2）　それを繰り返している間<は>，MSのいうことを誰も信じない
例１は，主語＋「は」＋述語，と考えられなくもないが，日本語には，主語は必ずしも必要ではないと考えて，「MSのスケジュール」という題目を提示している助詞とも考えられる。例２は，この方法でしか説明がつかない）

2）対比

例）　Internet Explorer<は>使うが，Outlook<は>使わない
普通に考えると，主語はユーザーですね。また，前述したように，「Internet Explorerは使う」と単文でいい切った場合，次の文章に変な期待がかかってきます。

3）限度

例）　Windows を使うには，３回<は>再インストールする覚悟が必要だ
少なくとも，という意味

4）再問題化（条件の保留）

例）　IE5は頑張っているように<は>見えた
「ただし，より不安定になった」のように付加条件を期待させる。
「が」：基本的に，上記１）に類似した使い方。ただし，
「は」：対応する動詞とある程度距離を許す
「が」：対応する動詞をすぐ直後に期待させる
例１）アメリオさん<は>アップルを再建したといっている
＝アメリオさんはいっている
例２）アメリオさん<が>アップルを再建したといっている
＝アメリオさんが再建した

お互い，逆に取れないこともないですが，一般的には上記のような感じを与えます。

私のプログラムとまったくあっているわけではありませんが，外れてもいない，といったところでしょうか。長い間，私も「は」と「が」の違いが気になっていたのですが，これで，迷ったときの判断基準を持てるようになりました。

たいへんなベストセラーになってしまって，本屋で買うのも気後れしそうですが，参考文献１は，おすすめです。

単語の順序

「調子は，いわゆる文章の音楽的要素でありますから，これこそは何よりも感覚の問題に属するのでありまして，言葉を以て説明するのに甚だ困難を覚えるのであります。つまり，文章道において，最も人に教え難いもの，その人の天性に依るところの多いものは，調子であろうと思われます」（谷崎　潤一郎「文章読本」）

そうはいっても，なんとかプログラムしなければいけないので，以下が私なりに考えた結果です。

●名詞の修飾語

たとえば，名詞をさまざまな語が修飾します。

例）　deux stylos rouge que j'ai acheté hier.
（two red pens that I bought yesterday.）

これを，仏語と同じ順に並べると，
「２本の赤い私が昨日買ったペン」
となります。「２本の」がどこにかかるのかがよくわからず，ちょっと落ち着きのない文章になってしまいます。そこで，数詞・形容詞よりも関係代名詞句を先に表示する，というルールを作りました。
「私が昨日買った２本の赤いペン」
となり，わかりやすい日本語になります。

そのほかにも，

例）　Les deux stylos rouge.
（The two red pens.）

も，
「それらの２本の赤いペン」
→「２本の赤いそれらのペン」
というように冠詞がなるべく名詞の近くになるように並べ替えます。このようなルールが細々とあるのですが，
関係代名詞
動詞

数詞
形容詞
疑問詞
冠詞
という順で，名詞に近くなるように並べ替えて表示します。実際の処理は，ファイル「print.cpp」中，関数PrintOutNoun()の，

SortToken(p->child, nounSortTable);

で行っています。nounSortTableは，上記のルールを収めたテーブルで同じファイル中にあります。

●副詞

同じ理由で，動詞に係る修飾語も，並べ替えを行っています。ファイル「print.cpp」中，PrintOutVerb()の，

SortToken(p->child, verbSortTable);

です。verbSortTableの中身は，

呼びかけ
感嘆詞
複文
前置詞の"Quand"
関係代名詞，副詞
前置詞の"En"
その他の前置詞
形容詞

の順に動詞に近づくようになっています。

また，修飾ではありませんが，主語をどこに持ってくるかも調整します。基本は，

主語 (副詞)(目的語1)(目的語2)動詞

ですが，主語と動詞が離れすぎると，対応がつきづらいので，

副詞句にTokenの数 >= 主語のTokenの数 + 3

のときには，

副詞1 "," 主語 副詞2(目的語1)(目的語2)動詞

という順序にします。すべての副詞を主語の前に移動すると，不自然な場合があるので，副詞2に，元の副詞Tokenをひとつだけ残して，残りを副詞1へ移動します。

例) Il mourut à Vienne le 26 mars 1827.
　　ウィーンで，1827年3月26日，彼は死んだ
→　ウィーンで，彼は1827年3月26日死んだ

●数詞

ある意味，名詞にかかる数詞は，形容詞と同じ扱いにしてもよいのですが，日本語では，動詞を修飾する語として扱ったほうが自然な場合があります。

例) J'ai trois frères. (I have three brothers.)
　　私には3人の兄弟がいる
→　私には兄弟が3人いる

●重文

基本的に，主語+目的語+動詞という順序で出力しますが，仏語が複文の場合，少し手を加えることで自然な日本語になります。

例) Je le dis. (I said it.)

は，「私がそれをいった」と訳されますが，

例) Je dis que elle est belle. (I said that she is beautiful.)

を「私は彼女が美しいといった」とすると，ちょっと落ち着きのない文章になってしまいます。ここで，日本語は助詞の使い方さえあっていれば，主語と目的語の位置を入れ替えることができる，というルールを思い出して，「彼女が美しいと，私はいった」と入れ替えます。

●複文

複文も，動詞Tokenの → child枝に接続されるという意味では，扱いは，副詞と同じです。主文の主語と動詞の間に入らないようにします。

劣等比較(より少なく)　moins(less)

これも日本語にはない考え方です。

例) Elle suis moins grand que moi.
　　(She is less tall than me.)

無理に訳すと「私より彼女は，より少なく背が高い」とでもなるのでしょうか，要は，「私ほど彼女は背が高くない」ということですね。

参考文献2によると，「形容詞の意味を反対にして訳してはならない。つまり『私は彼女より背が低い』ではない」とありますが，個人的には，両者の差をまったく感じないので，

劣等比較(moins) + 形容詞・副詞(grand)(less tall)

を見つけたら，両者を反対語にして，

比較(plus) + 形容詞・副詞の反対語(petit)(more little)
「私より彼女は背が低い」

と翻訳させています。

この反転を実現させるために，形容詞・副詞の辞書には，反意語を入れるコラムがあります。とはいっても，すべての形容詞・副詞が反意語を持つわけではないので，反意語が辞書に登録されていないときには，しょうがないので，

「より少なく」+形容詞・副詞

という不自然な日本語を表示します。

とりあえず，Windows版にしてみました

プログラムもこの規模になると，変数の初期化忘れなどの単純なプログラムミスというよりはアルゴリズム自身が抱える問題とか，ちょっとした定数の調整，というような作業が頻繁に発生します。

ちょっとした文章でしたら，DOSプロンプトに途中結果を表示して，いろいろ考えても問題ないのですが，いかんせん80×25文字の画面に収まる情報量というのは，たかが知れていますから，出力をファイルにリダイレクトしたり，Hard Copyを取ったりしていたのですが，とても非効率です。

というわけで，今回のCD-ROMには，Windowsに移植したプログラムを収録しました。翻訳プログラム自体は同一ですので，日本語訳の品質も同等なのですが，Windowsで走ることによって，途中結果のログがバックスクロールにより，すべて参照できるようになりました。

今回収録したプログラムには掲載されていませんが，いま私が行っている作業は，いかに日本語訳候補を減らすかという問題です。

たとえば，文章中に<de>という単語が5つあったとします。この単語は，冠詞/(動詞を修飾する)前置詞/(名詞を修飾する)前置詞/(動詞の原形を誘導する)前置詞……など，6種類の可能性がありますから(といっても，この数値はあまり正確ではない)，$6^5 = 7776$とおりの候補を検討し，いちばん得点の高かった訳を選ぶ，という作業を行わなければなりません。なかには，途中で，まったくナンセンスな組み合わせに対して，作業を行っていることに気づく場合も十分考えられます。

この問題は，人工知能のわりと初歩的なアルゴリズム(Min-Max狩り)なのですが，作業結果のログは，膨大なものになり，とても，DOSプロンプトとプリンターのコピーだけでは，やっていられません。

さらに，複数のフォントを混在できるというWindowsの利点も利用しました。仏語には，英語にない「è」とか「è」という記号があるのですが，これはシフトJISのコードに含まれていません。それぞれ，キャラクタコードとして，0xe9, 0xe8という値を持っているのですが，シフトJISでは，これらの値は別の漢字に割り当てられています。そういう理由で，表5のような代用記号をいままで使っていたのですが，Windows上で動作することによって，その必要がなくなりました。ただし，標準の109キーボードからこれらの文字を入力することはできませんので，入力にはいままでどおり，代用記号を用いてください。たとえば，「|」とキーボードから打つと，画面には「è」と表示されるはずです。

と，ここでWindows2000のβ版を見せていただいたのですが，なぁ〜んと，DOSプロンプトにも，スクロールバーがついていて，過去の出力結果が参照できるのでした(編注：Windows NTでは昔からできました)。Win

dowsにすると，画面表示がちょっと遅いので，ひょっとしたら，次回からは，DOS Promptベースのプログラムに戻すかもしれません。

そういうわけで，今回の収録プログラムは，前回DOSベースだったものが，Windowsに移植されてバックログが取れるようになった，という範囲です。辞書をどのディレクトリから参照するかとか，動作モードの通常/デバッグログ付きの切り替えなどの機能は，まだインプリメントしていません。次号が発売される頃には，私もWindows2000をインストールし，いろいろ実験しているでしょうから，その結果は次回，ということで(Visual StudioのWindows2000対応優待アップグレード9万円は痛かったなぁ……)。

参考文献

1) 大野　晋「日本語練習帳」1998　岩波書店
2) 倉田　清「仏文和訳の実際」1977　大修館書店

表3　活用形の拡張

ラベル	活用形	コメント
JP_KIND_GODAN ～ JP_KIND_GODAN_WA	ア行五段活用　～　ワ行五段活用	
JP_KIND_GODAN_GA	ガ行五段活用	
JP_KIND_GODAN_ZA	ザ行五段活用	
JP_KIND_GODAN_DA	ダ行五段活用	
JP_KIND_GODAN_BA	バ行五段活用	
JP_KIND_GODAN_KA2	カ行五段活用（その2）	連用形でツ音便化：行/った
JP_KIND_IRU ～　JP_KIND_WIRU	ア行上一段活用　～　ワ行上一段活用	
JP_KIND_GIRU	ガ行上一段活用	
JP_KIND_ZIRU	ザ行上一段活用	
JP_KIND_DIRU	ダ行上一段活用	
JP_KIND_BIRU	バ行上一段活用	
JP_KIND_ERU ～　JP_KIND_WERU	ア行下一段活用　～　ワ行下一段活用	
JP_KIND_GERU	ガ行下一段活用	
JP_KIND_ZERU	ザ行下一段活用	
JP_KIND_DERU	ダ行下一段活用	
JP_KIND_BERU	バ行下一段活用	
JP_KIND_SHIMOICHI2 ～　JP_KIND_SHIMOICHI2_WA	ア行下一段活用　～　ワ行下一段活用	名詞修飾の際，連体形よりも受け身にしたほうが自然なもの 失う記憶　→　失われた記憶
JP_KIND_SHIMOICHI2_GA	ガ行下一段活用	
JP_KIND_SHIMOICHI2_ZA	ザ行下一段活用	
JP_KIND_SHIMOICHI2_DA	ダ行下一段活用	
JP_KIND_SHIMOICHI2_BA	バ行下一段活用	
JP_KIND_NOROOT	語幹のない上/下一段活用	得る
JP_KIND_KAHEN	カ行変格活用	
JP_KIND_SAHEN	サ行変格活用	
JP_KIND_SAHEN2	サ行変格活用（その2）	名詞修飾の際，過去形にしたほうが自然なもの 怪我をする人→怪我をした人
JP_KIND_KEIYOSHI	形容詞（型）	
JP_KIND_KEIYODOSHI	形容動詞（型）	
JP_KIND_RENTAISHI	連体詞型形容動詞	活用は形容動詞とほぼ同一だが，名詞修飾の際の語尾が異なるあそこな店→あそこの店
JP_KIND_ADVERB	副詞型形容動詞	活用は形容動詞とほぼ同一だが，動詞修飾の際の語尾が異なる ゆっくりに歩く→ゆっくりと歩く
JP_KIND_ARU		在る（否定形：ない）
JP_KIND_MASU		ます
JP_KIND_RASII		らしい
JP_KIND_TA		た
JP_KIND_DA		だ
JP_KIND_DESU		です
JP_KIND_DEVOIR		表3参照しなければならない(否:する必要はない)
JP_KIND_FALLOIR		表3参照しなければならない(否:してはならない)
JP_KIND_POUVOIR		かもしれない(否：とは限らない)
JP_KIND_GODAN2 る～　JP_KIND_GODAN2_WA	ア行五段活用（その2） ～　ワ行五段活用（その2）	終止形・連体形では，進行形にしたほうが自然なもの 私は彼を知る　→　私は彼を知ってい ただし，否定形の場合は通常の五段活用を行う 私は彼を知っていない　→　私は彼を知らない
JP_KIND_GODAN2_GA	ガ行五段活用（その2）	同上
JP_KIND_GODAN2_ZA	ザ行五段活用（その2）	同上
JP_KIND_GODAN2_DA	ダ行五段活用（その2）	同上
JP_KIND_GODAN2_BA	バ行五段活用（その2）	同上

第103回

机の上で「鬼ごっこ」をするロボットたち

有田隆也 Arita Takaya

鉄腕アトムという夢

つい最近までは，あまり思ったこともなかったのですが，このところ，「これからロボットは面白い」と思うようになってきました。なにがどう面白いのか，それを今回は書きたいと思います。

いままで面白いとは思わなかったタイプのロボットのひとつの究極が，学生時代に自動車工場の見学に連れて行かれたときに見た，いわゆる産業用ロボットです。さまざまなロボットたちが恐ろしく従順に黙々と作業するシーンを見せてもらいました。多くの見学者は，たぶん，技術の進歩は素晴らしいとか，自分もこういうロボットを作ってみたいとか，やっぱりT社製の車は先進技術に支えられているのだなとか感心したのでしょうが，へそ曲がりの私は，2つのことを感じたのでした。

ひとつは，合理化というのも行き着くところまで行ったなあという感慨でした。ほとんどあらゆることにおいて，無駄というか冗長性は必要だと思うのですが，そこには，それが微塵も感じられなかったのです。

もうひとつの印象は，こんなのロボットでもなんでもないやということです。ただただ，それぞれの作業に特化した製造マシンにすぎないと感じたのです。やはりロボットというからには，我々の抱く「ロボット」のイメージに近い存在であってほしいという気持ちがあるわけです。つまり擬人化されたイメージです。

先日，評論家の立花隆氏が，日本は今後「ロボット」の分野で世界をリードするだろうと述べていました。もちろん，自動車製造マシンとしてのロボットとは対極的な生命らしさを感じるようなロボットをここでは意味しています。その理由として，彼は，手塚治虫の漫画「鉄腕アトム」の存在を挙げていました。ロボットを作っている，あるいは作ろうとしている日本の多くのロボット研究者の頭の中には，感情移入することのできるあの鉄腕アトムのイメージが刷り込まれており，このことこそが，外国のロボットを作る人たちとは根本的に違っており，生命らしさを持つロボットの実現に向けて重要なのだという指摘です。

マニアックなロボット

ペットロボット，あるいはデジタルロボットという言葉がよく聞かれるようになってきました。25万円というやけに高価な（しかしお金があれば買ってしまいそうで，実際にすぐ売り切れてしまう）ロボット犬「AIBO」を開発したソニーは本腰を入れて取り組んでいる様子です。6種類のセンサが内蔵された，どこが可愛いのか私にはわからないおしゃべりペット「ファービー」も入ってきました。

ところで，一般に，機械を自律的に動かす（人間から逐一指示が与えられなくても機械が勝手に判断して自発的な行動を取る）ことに対するお堅い理屈づけとしては，次の2つがあるとのことです（文献1を一部参考にしてます）。

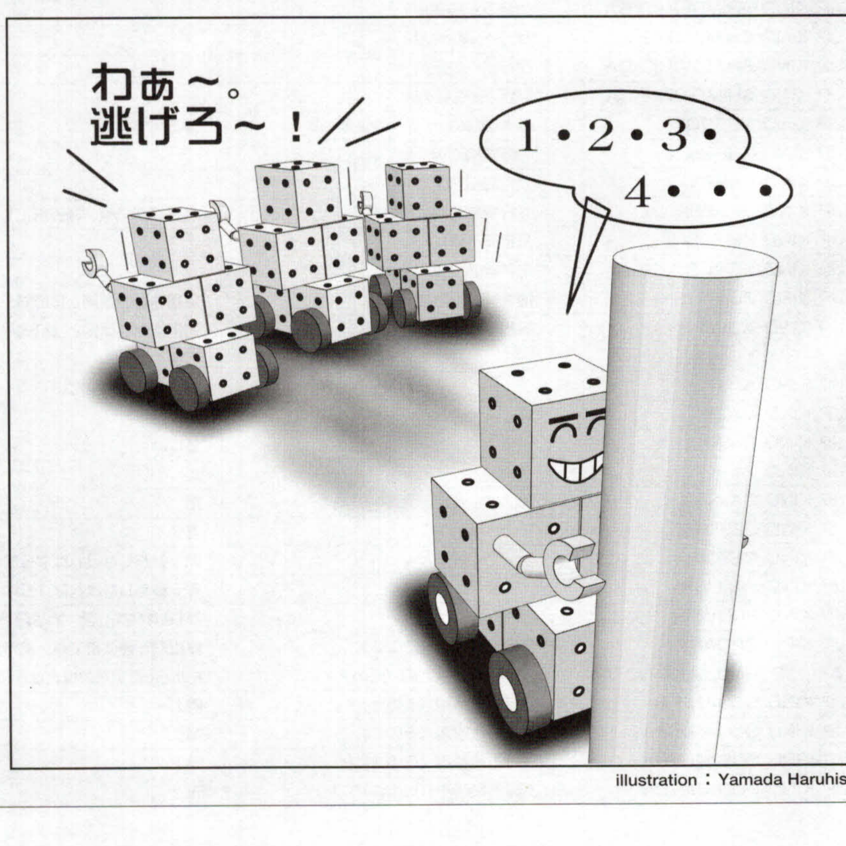
1） 効率的側面

機械（コンピュータ）に対する指示のしかたとして，現在主流なのは「あーせい，こーせい」と直接的具体的に指示する「直接操作」と呼ばれるものだが，指示する内容が高度化複雑化するにつれて，それが簡単でなくなる。そこで，あまり細かなことまでくどくど説明しなくても，自律的にやってくれる「間接管理」が必要になる。

2） 心理的側面

擬人化された機械のほうが人間は指示を出しやすいし，機械がどのように行動するかが予測しやすい。また，人間の心が癒されるという面もある。そもそも，人間は，自律的に動く人工物に対して，それが合理的な意図を持つ主体と判断したがる傾向があるのだから，それを使おう。

1）は，要するに機械と人間のインタフェイ



illustration：Yamada Haruhisa

ス技術の発展形として考えましょうということです。私はこの1)については，研究の位置付けというか，建前的，あるいは世間体的には必要で重要なものであろうとは思う一方で，かなり懐疑的であります。ひとつは，あまりに簡単な処理について，人モドキが出てきてなんだかんだとうるさくつきまとうようなインタフェイスはこりごりということがあるからです。これをしろというボタンがひとつあればいいのにと思われたことが皆さんにもあるでしょう。

一方で，自律的に振る舞う機械に複雑な処理を任せるということにも，かなりの抵抗感があります。エアコンの温度調節を任せるというのとは話が違います。他人にものを頼む場合でもそうではないでしょうか？　その人がどういう人間で，どういう場合にどのように振る舞ってということをある程度把握してからでないと不安ですよね。よっぽどの信頼感が形成されていないと大切なことは頼む気がしません。この問題はたぶん技術の進歩がそのうち解決してくれるとは思うのですが。

私としては，2)の方向性というか，とにかく，鉄腕アトム追求，趣味性追求でいきたいと思っています。機械の効率性が，本当に機械に自立性，あるいは生命性を付加することによって，直接的に向上するか，いまのところ確信が持てませんので。2)は，人間側に視点を置いた立場です。どういうロボットの振る舞いのなかに生命性を我々が感じ，また，結局，そのようなロボットが我々にどのような影響を与えるのかということに興味があります。

現在，開発されている（趣味的な）ロボットに関しては，

http://findx.nikkeibp.co.jp/robot_0.html

にリンク集があります（産業用のロボットも含まれてますけれど）。ソニーのAIBOと，LEGO社のMINDSTORMSは別格とし，それ以外にも20ぐらいのロボットがリストアップされています。

趣味的なロボットとひと言でいってもいろいろと性格が分かれます。生命性というものを最初から振る舞いのなかにすぐ表れるようにしようとすると，感情とか，知能とか，そういうものを最初から入れ込むということにつながりやすくなります。ソニーのAIBOも，今後，ユーザーがさまざまなレベルで自由にいじれるようになっていく方向にあるのだと思いますが，現状は，基本的には，製作者側が仕組んだ仕掛けを楽しむというのが，基本的なところではないでしょうか？（まだ，触ったことがないので，確実ではありませんが）

ここでは詳しく説明しませんが，生命性の本質は，「嫌なことをされたら怒って機嫌が悪くなる」というところにあるのではなく，「嫌なことをされたら怒って機嫌が悪くなるようになっていく」というところにあると思います。つまり，学習や進化などを通じて，次第に適応していくメカニズムが備わっているかどうかという点です。結果としての振る舞いは，いわば二次的なところであると考えます。

ところで，コンピュータのユーザーにもいろいろなレベルがあります。たとえば，市販のアプリケーションだけを使うというのでも，十分役に立ち，また十分楽しめますが，もっともっと楽しもうとするならば，やはり，自分でコンピュータを組み立てたり，プログラミングしたりすることに発展するのは自然なことでしょう。同様に，ロボットにおいても，深いレベルの喜びとでもいうのでしょうか，そのようなものを味わうためには，自分で組み立てたり，自分でその機能や動作をプログラミングできたりすることがかなり重要なことだと思います？

自由にロボットの構造を作ることができ，しかも，その動作を自由にプログラミングできる代表的な製品は現在次の2つだといえるでしょう。

1）LEGO社のMINDSTORMS

8ビットプロセッサの入ったブロックRCXをはじめとして，モーター，センサ（光センサ，タッチセンサ），各種部品（タイヤ，軸，歯車，コード，装飾用部品，普通のブロック）など部品数700のセット。コンピュータからRCXにプログラムやデータを転送するための赤外線送受信器IR Tower，制御用のビジュアル言語の入ったCDも付属。1セット35,000円程度。もう少し安めで，コンピュータを使わないで本体の設定だけで簡単な動作の設定のできるタイプも発売。

2）システムワット社のROBOCUBE

一辺5cmの立方体のブロックを組み合わせてロボットを作る。それぞれのブロックはセンサやモーターなどさまざまな機能を持つ。独自のビジュアル言語が付属しており，パソコンでプログラムを作成し，ロボットにダウンロードして，自律的に動作させる。1セット22万円。カラフルなブロックが楽しい。

私の研究室（人工生命ラボ）で行われている研究のほとんどは，簡単にいうならば，コンピュータでプログラムをがんがん動かして，作った本人でさえびっくりするような結果を出したいというタイプのものです。そのような研究にどっぷり浸かっていると，目の前でロボットを実際に動かすということがすごく新鮮で楽しくなってきます。コンピュータの中の世界と現実の世界を結びつける手がかりになるという点に，学問的な意義ももちろんありましょう。そういうことで，ロボットがらみの研究を少しずつ始めようとしている段階です（たとえば「ロボット間のコミュニケーション」というテーマには面白いものがたくさんあると感じています）。

デスクトップ型ロボット研究環境

1)のMINDSTORMSは1年前に本連載でも取り上げました（文献2）。これを2セット使って研究室でやっていることを紹介したいと思います。本邦初公開です。また，2)のほうも，これからのパーソナルなロボットの可能性をさらにいろいろと試していくうえで面白いのではないかと思い，現在，試し始めています。これについても，少しだけですが紹介しましょう。

とりあえずの目標というか研究のモチベーションは，好きなロボットを自由に組み立て，しかも，最初はまったく無能でも，学習や進化によってだんだんと賢くなっていくような楽しいロボット実験をロボット工学の専門家でなくても誰もが簡単にできる環境を作りたいということでした。要するに，お金がかからずに，し

図1　ロボットとコンピュータにおけるマニアック度

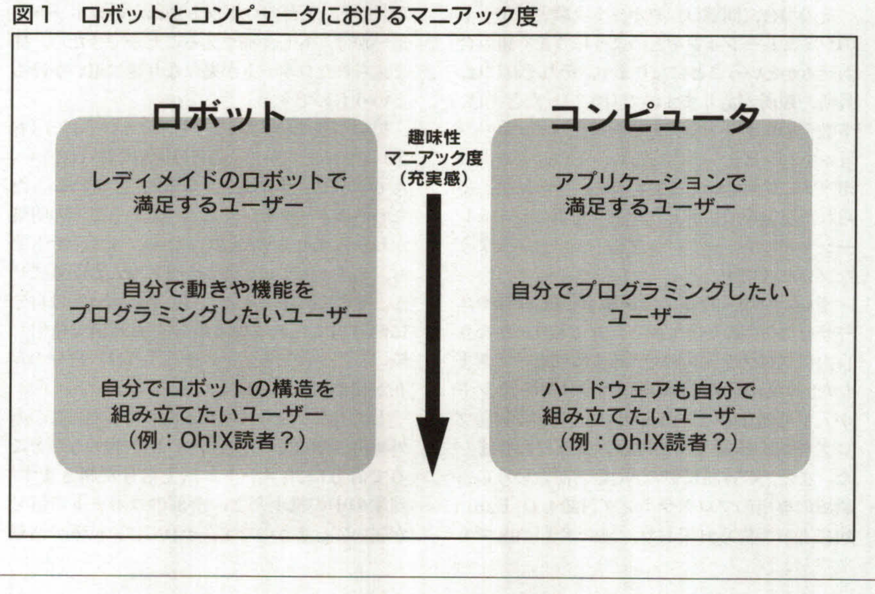
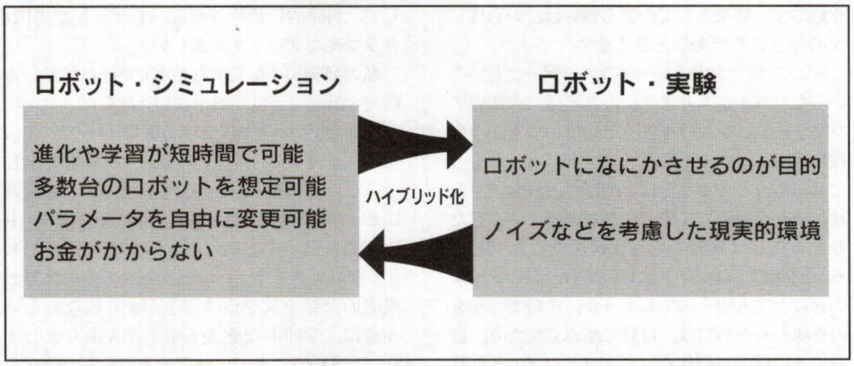


図2　シミュレーションと実験の融合環境



かも，難易度的にもそう難しくなく，机の上で気軽に楽しく！　という環境です。

MINDSTORMSはこの目標をすでにかなりの程度まで実現している製品であるということができます。これが登場する以前は，それこそロボットや機械工学の専門家でないとこのようなロボットを使って自由に遊ぶことはできなかったといっていいでしょう。

ただし，単に1台のロボットを使って，「ものにぶつかったり，床に書いてある黒い線をまたぎそうになったら，あとずさりして向きを変える」程度のプログラムならば，付属のビジュアル言語で簡単に作ることができますが，より複雑なことをさせようとすると，限界がすぐに見えます。さらに，生命性や自律性を追求するために，学習や進化のメカニズムも組み込みたいということになると，より高いレベルのプログラミング言語を使ってプログラミングする必要性が生じますし，また，ロボットにいきなり制御プログラムをダウンロードせずに，コンピュータでシミュレーションしなければならなくなります。

その場合，問題は，ロボット実験とロボットのシミュレーションをどのようにうまく組み合わせるかということになります。それぞれには，長所，短所がありますので（図2）。たとえば，多数台のロボットを想定した進化シミュレーションをコンピュータ内で行い，その結果を，ロボットに反映させて実験を行い，その結果，得られた現実環境に即したデータを再びシミュレーションにフィードバックし……，というようなプロセスです。

幸い，ソフトウェアの環境もLEGO社やユーザーたちのおかげで揃ってきており，かなり自由にプログラミングできるようになってきました。いわゆるOCXの形式でコンポーネントが入手可能なので，VisualBasicなどで制御プログラムを作ることができるようになりました。また，C言語に似たNQC言語というC言語風の専用のプログラミング言語もD. Baum氏によって作られており，MacやLinuxでも利用できるようになっています。さらに，RCX Command Centerという，センサなどの制御や状態表示をリアルタイムに行ったりNQCのコンパイルや転送を容易にする環境もあります。

私の研究室の学部生である竹内君は，シミュレーションにはJava言語を用い，ロボットのプログラミングにはNQC言語を用い，ロボットとコンピュータのインタフェイス部分にはRCX Command Centerを使っています。同一のプログラムでシミュレーションもロボットの制御プログラムも書けるならば，本当の意味で，実験とシミュレーションのハイブリッド化が実現されたといえるのでしょうが，いまはこうなっています。

ロボットたちの「鬼ごっこ」

さて，シミュレーションと実験が統合化された安価でお手軽なデスクトップ環境でもこういうことができますよという実例を示そうということで選んだ題材が，ロボットたちの「鬼ごっこ」です。1台のロボットがほかのロボットを追いかけ，もし，捕まえることができたら，捕まえられたロボットが鬼になり逆に追いかけるというものです。

思いつきで「鬼ごっこ」を作れというほう（有田）は簡単ですが，やるほう（竹内君）はたいへんです。センサ機能が，基本的に，ぶつかったらわかるタッチセンサとごく近いところの明暗がわかる光センサだけですから，まず，そもそも，相手がどこにいるかがぜんぜんわかりません。そこで，工夫1として，各ロボットは自分に割り当てられている番号を赤外線で発信し続けることにしました。そうすれば，自分の存在が相手に伝わるはずです。

ところが，さらに困難がありました。この赤外線は，測ってみたところ，障害物のないところで前方に20メートル以上信号が届きます。部屋の中の机上だと，全部のロボットに信号が届いてしまうのです。それでは，位置の情報がないことと同じです。そこで，工夫2として，赤外線発信部の正面のところを0.5cmだけ開けて，その左右を紙で覆うことにしました。これでやっと指向性が出たので，ロボットが回転して信号の聞こえるほうに向かうということができるようになりました。

また，鬼のロボットが別のロボットを捕まえたとしても，捕まえられたロボットが捕まえられたとわからなければ，鬼の交代ができません。そこで，捕まえたときには捕まえたという信号を送ることにしました（子供がやる鬼ごっこでも，「タッチ」とか「捕まえた」ぐらいいいませんでしたっけ？）。あとは，ロボットたちが黒い線から出ないようにする（付録の紙を使う）ことで，鬼ごっこをするだいたいの方針はできました。

さて，これで鬼ごっこもどきをするようにプログラムを作って完成！　では，あまり面白くありません。最初はわけもわからずランダムに動いているが，捕まえたときに「よくやった」とほめてやることにより，だんだんと相手を追いかけるように学習させたいということで，竹内君の格闘は続きました。そこで，学習のメカニズムとして，ここのところ比較的注目を集めている「強化学習」を利用することにしました。これは，ああして，こうしてと，直接的に答えを教えるというタイプの学習ではなく，行動していった結果，うまくいった場合にうまくいきましたよという報酬だけが環境から与えられ，それによって少しずつ学習させようというものです。

この研究では，ロボット2台で1対1の鬼ごっこをさせる場合には，鬼ロボットにおいて，相手ロボットを発見しているかどうか，相手に接触しているかどうかという2×2＝計4通りの状態を考え，それぞれの状態において，回転するか，直進するか，捕獲するか，後退するかという4つの行動のうちどれを選択するかと評価値を少しずつ更新することを行いました。評価値の大きさに比例して，確率的に行動を選びます。

また，うまく相手を見つけた場合には，捕獲直前の行動から3ステップ前までの行動に報酬を（減少させながら）与えます。これは，いわゆる「利益共有法」という報酬の与え方に基づく学習です。

実ロボットで試行を繰り返して学習させていくのは，とても手間ひまがかかりますので，こういう場合こそ，シミュレーションのほうで行います。そして，結果をロボット実験の制御プログラムに反映して，本当にうまく追いかけるようになったか確認します（確認だけでなく，シミュレーションのほうに物理的制約に基づく影響を再び反映させるといいのですが，そこまでにはまだ至っていません）。

ほかにも，複数台のロボットが協調して1台のロボットを追いかける鬼ごっこなども行いま

図3　鬼ごっこをする2台のロボット

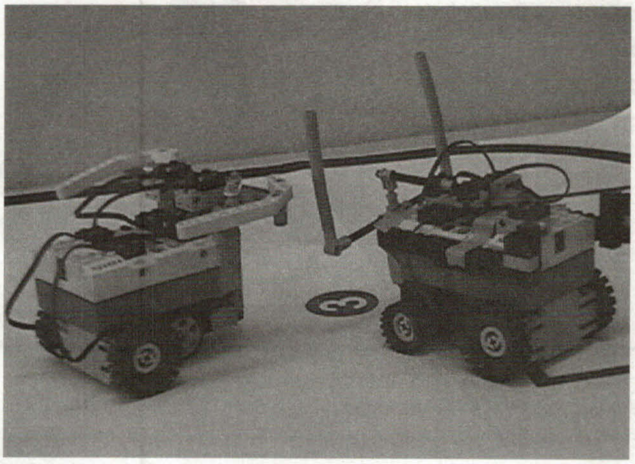
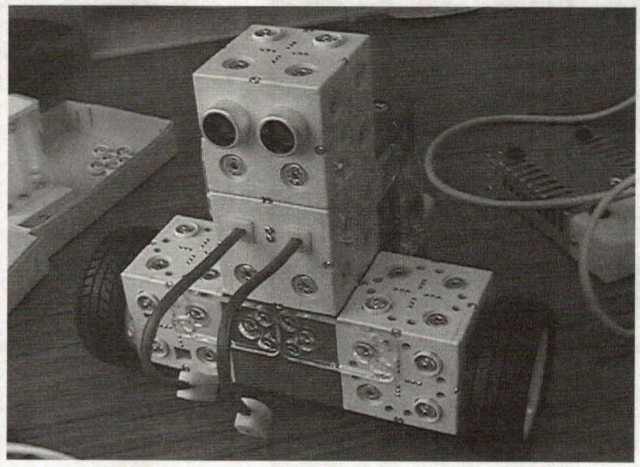


図4　ROBOCUBE組み立て例



した。いずれにせよ，だんだんと賢くなるという鬼ごっこロボットの実験がほんの数万円のロボットシステム2つとパソコンを使ってできたということです。

学習前と学習後のMINDSTORMSロボットの「鬼ごっこ」の一部を撮ったムービーも含まれたプレゼンのページを用意しておきますので，興味のある人は私の研究室のホームページから見てください（現在は，IEからしか見えないようなのですが，そのうちほかのブラウザでも見えるようにしたいと思っています）。人工生命ラボのホームページは以下のとおりです。

http://www2.create.human.nagoya-u.ac.jp/

ROBOCUBEと吹き出すお姉さま

さて，マニアックなロボットとして，もう1段階進むとROBOCUBEということになりましょうか。このROBOCUBEは小さなベンチャー系企業が作っているということもあって，いろいろと面白いことがあります（いちばん面白いことは最後に書きます……）。たとえば，ブロックとブロックをつなぐ場合，ブロックの面同士を4箇所でジョイントボタンを挟んでくっつけるのですが，その接合する金具として，洋服に使われている，いわゆる「ホック」が使われているのです。ブロックの面側にはホックのメス側が埋め込まれており，オスとオスの背中をくっつけたジョイントボタンを間に挟み込むというものです。アイデアですよね。

シリアル信号，および電源はこのホック経由で伝わるのではなく，別に接点が設けてあり，どのような向きにブロック同士をくっつけても正しく接続されるようになっています。各ブロックにはプロセッサが内蔵されており，接続すると自動的に信号線をバスとするLANが形成されるようになっているのです。

センサやアクチュエータはMINDSTORMSよりはずっと本格的なものとなっています。たとえば，超音波センサが含まれていて，障害物までの距離を3mまで正確に測ることができました。また，モーターはMINDSTORMSのような迅速な動きこそできませんが，DCサーボ機構があるので速度や回転角度を正確にコントロールできます。実際，その回転量によって現在位置がある程度の精度でわかるようです。タッチセンサも，押されたことだけでなく向きまでわかりますし，ライトブロックは好みの色をさまざまな光らせ方で光らせることができます。

プログラミングに関しては「タイル言語」という名のオリジナルのビジュアル言語が用意されています。少し使ってみた感触としては，思ったより本格的な作りだなあという印象でした。ブロックを好きなように組み合わせると，LANを通じて自動的にブロック構成の情報がコンピュータに伝達されます。

さらに，こちらもOCXが提供されており（オプション），高レベルの言語でプログラミングすることができます。ただし，タイル言語を使わないで，高水準言語を用いる場合は，プログラムをダウンロードして自律的に動作させることはできず，パソコンからロボットをコントロールする形式になります。これに関しては発売される（た？）赤外線通信ブロックを購入すれば，無線によって外見上は自律的に動作させることができます。

ネットワーク，画像処理，制御システムなどの業務に携わってきた小さな会社が「夢を実現する」（ホームページより）という理念を実現するために世の中に送り出した，この一見使用目的不明ですが（教育用にはいいでしょう），だからこそぜいたくといえるこのロボットをいかに使うか，それに関しては，いま，他分野の研究者の方ともいろいろと相談しながら構想を練っている最中です。そのうち，なにか面白いものができたら皆さんにも見てもらいたいと思います。

なお，ROBOCUBEの情報は以下のホームページに載っています。

http://www.watt.co.jp/

さて，忘れてはいけませんね。この商品のもっともユニークなところはどこかというと，商品コンセプトとか，技術とかいうよりは，なんといっても，添付されてくるCD-ROMにあるのです。CD-ROMの中には，ビジュアル言語ソフトウェアだけでなく，ナレーション入りの使用ガイドが入っています。

説明してくださるお姉さんは，出だしからしばらくの間は調子よく，てきぱきと的確に説明してくださる（雇われて読んでいるのではなく，意味がわかって読んでいるという感じ）のですが，半分ぐらい読み上げ仕事が終わったころでしょうか，いくつかの固有名詞の発声に数回つまずいたところで，あまりの自らの不甲斐なさに思わず笑えてしまわれたのでございましょう。「ぷっ」とはっきり吹き出されたのでございます（私もつられてしまいました）。20数万円という高価な商品に付属する説明ナレーションで思わず吹き出しているというのは，古今東西を見渡しましても空前絶後のことではないかと推察いたします。まあ，とにかく，面白いロボットです。

参考文献）
[1] 牛田博英，平山裕司，中嶋宏，「デジタルペット：心をもった機械達」，情報処理，Vol. 41, No. 2, pp. 127-136, 2000.
[2] 有田隆也，「知能機械概論第101回：Papert博士からの2つの贈りもの」，Oh!X，1999年春号，1999
[3] 竹内康貴，「実験とシミュレーションを融合したデスクトップ型ロボット研究環境」，1999年度名古屋大学情報分化学部卒業論文，2000

STUDIO X ON AIR

FROM READERS TO THE EDITOR

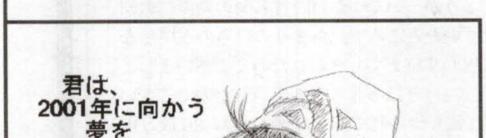
予定の号をひとつすっ飛ばしてまたまた春号です。またまた危険な進行で，この切羽詰まった時期にひたすらはがきの打ち込みをやっております。次からはバイトを雇いましょうか……。



◆PlayStation2にはかなり期待しています。某64ビット機が実はチップ間では66MHz PCIに近い状態でつながれていたり，ゲーム機というよりはCGA作成マシンに近い設計がなされていたのに対して，かなりゲームのためのコンピュータになっていると思います。配線の量は考えるだけでも気を失いそうですし，基板そのものが巨大なLSIといっていいかも。EmotionEngineやGraphicSynthesizerは文字どおり3DCGAのレンダリングやエモーション作成に専念させ，基本的なゲーム進行はPlayStation CPUが行うような気がします。やはり1/60秒にはこだわってほしいですしね。話は変わりますが，iBookはとてもいいです。取っ手がついて持ちやすいですし。大きくて重いという人もいますが，ムービングモバイルコンピュータKODOMOはどうなるんですか？　うまれてすぐ3kg，1年もたたないうちに10kgを超えてしまうんですよ。ほら，もう長男が膝の上にきた（3人目のラップトップ）。

伊藤義幸（41）愛知県 X68000

PS2ってそんなにゲーム機っぽいですかね？　I/Oプロセッサがメイン処理をするという手もなくはないでしょうが，オーバーヘッドがどれくらいかわからないのと，RAMが少ないという致命的な問題がありますからねぇ……。あと，iBookをお使いですか。自走式ならラップトップでも重くてもいいんですが，持ち歩くとなるとやはり辛いと思うんですが（長期間片手式やショルダー式バッグ使っていると背骨が曲がるそうで）。3kg超えたら取っ手ではなくてリュック式でないと……。



▲SHIMSOFT　広島県

◆1999年春号の「言語処理プログラムを作る　第1回」のおかげで，非線形最小二乗近似のプログラムを作ることがようやくできました。ありがとうございます。掲載後すぐに私が役に立てたことは昔からなかったのですが，あとから読み返しても役に立つような記事を期待しています。

小林豊（29）兵庫県 AT互換機

意外なところで役に立っているようですね。これからもいろいろな分野の記事を載せていくと思いますので，隅々までチェックしてやってください。

◆こんにちは，倉橋です。要望なのですが，是非，Oh!X休刊前のバックナンバーをCD-ROMで縮小版で出してほしいな，と思うのですが，如何でしょうか？　よろしくお願いします。

倉橋賢治（30）神奈川県 X68000 EXPERT II

著者とか連絡つかない人も多いので，古いものはちょっと難しいですね。手元に置くスペースもないんで電子化されると便利なのは確かなんですけど。

◆PentiumIII/450MHzのDOS/V機を買い，ついにインターネットを始めることになりました。これが初めてのe-mailです。いくつかホームページを，のぞいているうちにOh!Xはまだあるだろうかと思い，昨年買ったOh!Xを開き，おそるおそるアクセスしてみたところ，つながったためほっとしました。思えば，初めてパソコンを手にしたのがMZ-700でそのときからOh!X（当時はOh!MZだった）と，つきあってきました。山のようにあった本もいまではありません（実家には古いOh!MZが，いまだにある）。わたしのX68000はというと，甥っ子がパソコンをしたい，2台あるんだったらちょうだいといわれ，ついに手放してしまいました。甥っ子も幸せな人生を送ることだろう……。これからも応援します，がんばってください。

中島陸奥（32）鹿児島県 AT互換機

ということで，メールでSTUDIO Xに応募していただきました。初メールですか。自分相手以外の初メールなんてもう忘れてしまいましたけど，メールとWebの使い方がわかってくると手にできる情報量が飛躍的に増大します。ぜひインターネットを使いこなしてみてください。

◆Windows用MZ-700エミュレータには驚きましたが，もとにしたものがLinux用のMZ-700エミュレータということで，こんなものがすでにあるとは知らなかったのでさらに驚きました。MZ-700の実機を入手したくなりました。VMWareでWindowsを走らせる場合はWindows自身も必要ですが，WindowsNTの場合も必要なんですかね？　APIの働きだけを公開資料に基づいてエミュレートしているなら（この前提が間違っているかもしれませんが），Windowsなしでエミュレートしても問題ないですよね。

湯浅夏樹（32）奈良県 X68/040turbo，AT互換機

VMwareはPCのハードウェアをエミュレートするソフトなので，その上で動くソフトは必須となります。WindowsAPIをエミュレートしているわけではありません。そんなことができるなら，WindowsクローンOSやフリーWindowsがあふれまくっていることでしょう。WindowsAPIをエミュレートするくらいなら，PCのハードをエミュレートしたほうがまだ簡単ということでしょうね。MZ-700のほうは，コンパチBIOSを作ってしまえば，フリーで流せるのですが，本当に誰かやりません？

◆Windowsの開発環境は高い。高すぎる小説書きの

システムを組みたいのに。マクロにForth載せるとか……。　高田英基(27)千葉県 X68030, PC-P32X3

探せばフリーの開発環境もあるものですよ。とはいえ，お望みのものは次にあるMeadowでできそうな気もしますけど。要するに，Forthではなくて EmacsLispでやることになるわけですね。

◆WindowsマシンにMeadowをインストールしました。BFDフォントファイルをX68000の16ドットフォントにして使用しています。X680x0から16ドットフォントを吸い出し，X Windowや前述のMeadowで使うという計画は前からありましたが，そろそろ公開しようかと思いドキュメントを整備しているところです。　三津田哲雄(24)山口県 PC-9801

Emacs系のエディタは標準エディタのED系と並んで，X68000では人気の環境でしたね。ED風Micro Emacsとかも懐かしい感じ。WinMuleから最近はMeadowですか。どの環境でも同じエディタでいけるというのは強みですね。でも，フォントファイルの扱いにはくれぐれもご注意を。

◆自分はX68000を触ったことすらありませんが，内容が面白そうだったのでつい買ってしまいました。買ったあとに知ったのですが，特集記事のEasy Link LibraryのHPはいつもお世話になっています。でも，本当に付属のCD-ROMもX68000専用なんですね。　吉永康之(35)福岡県

はて？　CD-ROMは別にX68000専用というわけではないのですけど。誌面も一応，機種非限定ということでやってますし，どちらかというとWindows寄りだと思うのですけど。

◆本誌を懐かしく思い手にしました。「特別企画」を見てすぐに買いました。PICマイコンを最近使っていますので，大変役に立ちます。第5章のオリジナルエアチェックタイマの回路図が見つけられません(私だけかな?)。　小林誠(44)新潟県

すみません。回路図は掲載漏れでした。今号の　ページをご覧ください。PICマイコンはちょっとした工作では非常に人気の高いデバイスですね。応用例ができたらぜひ投稿してください。

◆広く深くじゃなくてバラバラだけどとても深くですね。ところでいつぞやのシャープからの新Xとあった編集後記ってあれはなんだったのですか？
大矢英光(26)新潟県 X68000SUPER

プレジデント誌に掲載されていたシャープの社長のコメントについてですね。まあ，そのときは開発していたんだということで理解しておきましょう。

◆Oh!MZ1984年7月号(だったはず)のX1用BATMANをはじめ，いろんな雑誌のプログラムを打ち込んでいました。弟にダンプリストを読んでもらいながら入力していた頃が思い出されます。そんな感じでX1Ckに親しんだ身としては祝一平氏の「試験に出るX1」は最高の連載でした。春号のmicroOdysseyを読んでから，昔のOh!MZを引っ張り出そうとしたら行方不明になっており，悔しい思いをしています。で，それをきっかけとして，プログラミング魂が7，8年ぶりに戻ってきています。VC++から始めるつもりですが，その前にPC本体を買わないと。学生時代からのCの本を読みながら記憶を掘り返しています。Oh!Xも買い続けますので頑張ってください。私もそのうち参加したいです。　菊池淳悦(29)岩手県

祝氏が逝ってもうじき1年。「なんで『祝一平』という名前なんですか」と聞くと，『祝』という青林檎知りませんか？　私はあれが大好きなんです。で，祝がいっぱい食べれたらなと……」ということで，以外と単純なペンネームでした。Macは少し使ってたけどApple信者ではなかったです。

◆よく見たらシャープの広告がないですね。広告の数も少ないし。あまりほかでやってない記事があるのはいいです。月刊化を目指して頑張ってください。最近インテルとAMDの戦いが面白いので記事にしてみたらどうでしょうか。そういうものはDOS/Vmagazineでやってるのかな。DOS/Vmagazineは見ていないのでわかりませんけど。
山口明宏(27)東京都品川区 AT互換機

とりあえず，シャープからは取ってこいといったんですけどねぇ。営業すらしてないでしょうね。困ったもんですねまったく。インテルとAMDの戦いはご想像のとおり，/Vmagでしょっちゅうやってる話題です。そちらを参照してください。それから，Oh!Xの月刊化はとても無理です。たまに，実は月刊化したほうが楽じゃないかと思うこともありますけど……。

◆インターネットでライブカメラというものを目にするようになりました。それには撮影転送用のソフトを使うようですが，割と重く，余っている古いパソコンを専用に使おうとしてもうまくいきません。とりあえず，一定時間ごとの撮影だけのソフトでも作ればよいかと思いますが，TWAINの使い方がわかりません。BASIC(できればF-BASIC)からTWAINなどでカメラ撮影を行う方法はあるのでしょうか。Windowsのプログラムはほとんど作ったこともないので，なにを見ればよいのかも教えていただければありがたいです。
横堀正敏(35)埼玉県 Libretto60, HP100LX

そりゃ，インターネットでしょう。http://www.vector.co.jp/あたりで調べればなにか出てくるものです。ざっと見ると，ezScan.ocxというのがそれっぽいですね。VBからのデジカメコントロールサンプルはついていますので。触ったことないんですけど，F-BASICってOCXの制御はできますか？

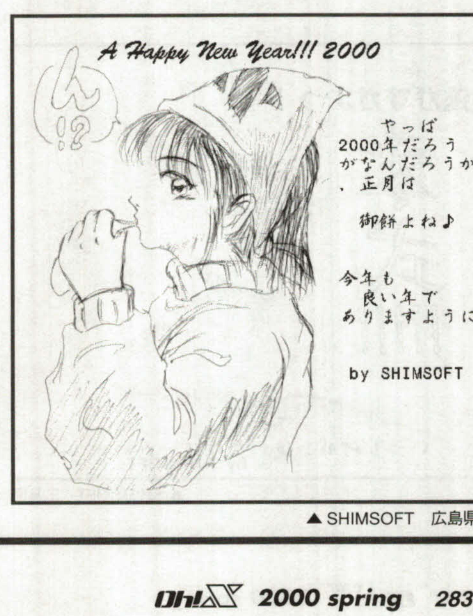
◆もっと幅広く読まれるように定価を下げたとのことですが，この本の厚さと内容からしてちょっと苦しいのではないかと思います。2500円でCD-ROM 1枚付きまで，このくらいのページ数だったらちょうどいいのではないでしょうか。だいたい音楽CD 1枚分くらいの値段がよいと思います。
成木出哉(22)滋賀県　X68000XVI

値段はなんとかなる範囲で設定されてますので気にしなくても大丈夫。できあがるまで全体像が見えないので，値段決めるのも大変です。当分はこんな値段とこんな厚さでいくことになるのではないかと思います。場合によってはかなり違ってくることもあるかもしれませんけど。

◆PCでマンガを描いてみるというのはいつか実現できたら素晴らしい。マンガが描けるならさまざまな図面も自由自在だ。気楽にデジタルデータにできると思う。「ペンで紙に描く」という感覚に近い入力デバイスはまだ未発達の分野だと思う。　五十嵐豊(32)神奈川県

なかなかどうしてタブレットは慣れるとそれなりに近い感覚で描けるようになるという話です。とはいえ，手元で描いたものが画面に出るというのでは取っつきやすいとはいえませんから，タッチスクリーンとタブレットの複合型というのが本命なんでしょうかね。

◆Oh!Xの古い号をよく見ます。難しいプログラムも優しく書いてあり助かります。Photoshopのエンボスフィルタの理論とプログラムとかをやってください。
安澤光男(49)神奈川県 X68030＋060turbo



▲SHIMSOFT　広島県

前号でもちょっとやってませんでしたっけ？　ちょっと違いますか。古いOh!Xをひっくり返すといくつか出てくるはずです。ずいぶん前からやってた話なんですけどね。Photoshopにエンボスがつくより前だと思います。EX-Systemのプラグインのソースが参考になるでしょう。

◆ひそかに残念だったのは須藤氏の記事がなかったことです。とはいえ，全体としてはレベル，怪しさともに向上し，昔のOh!Xのようでうれしい限りです。ということで，昔のOh!Xのように半分も読めずに本棚に並べています。少し色あせたOh!Xと成長したOh!X，これからも死人が出ない程度にがんばってください。

田辺敦司(24)東京都江戸川区 X68000ACE，AT互換機

とりあえず倒れない程度にやっております。さすがに体力が落ちてきているなあというのは自覚してますが。周りの人はみんな応援はしてくれるのですが，誰も心配してくれないんで寂しい思いをしています。普通は無理な話なんですけどね……。

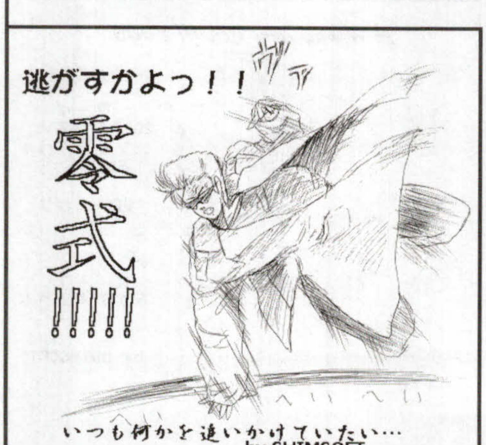
◆次はDolphinのハード特集でしょうか？　(でも情報ないですね)　是枝浩行(39)神奈川県 PC-DJ50M

まったくです。タレコミ募集中。DC2関連でも可。

◆前号より買ってみて思っていること。自分の知らなかった世界がいっぱいあって楽しい。Xのユーザーは凄いですね(特集の「未定」も凄いが)。でもこういうのも好きなので次号も買うかも。

高野勝志(24)新潟県 PC-9821

今回も買っていただけたでしょうか？　最近のPC界なんて話題が限られてますので，こういう本は珍しいかもしれません，というかほかにはないですね。実際のところ。本来，パソコンってもっといろんな用途に使えるものですので，この本では独自の方向でずっといきます。どうぞよろしく。



▲SHIMSOFT　広島県

◆読んで思ったのは「知りたいことが載っているな」ということでした。特にそう思った記事はDHTMLとPICを扱った記事です。現在，主にWindowsｍｐプログラミングをしています。ActiveControlやCOM，ActiveDocumentやNetscape Plug-inなど，さまざまなテクノロジを取り扱えるまで勉強していますが，それでもなお(目的のために)「あ，これも知ってなければ作れないな」という分野があります。上記の記事がそうでした。「必要」なテーマが載っていてとても役立ちます。これからも期待しています。

平井秀憲(26)千葉県 AT互換機

基本的な部分から解説しなおしている感じなので，ある程度のスキルを持った人にはもの足りないかもしれないなあと思っていましたが，お役に立ってなによりです。今度は書く側に回ってみませんか？

◆7月11日に会社が倒産して，ちょっと時間ができたのでマンガを描いていたりUNIX(X Window)のペイントソフトを作ってたりしています。GIMPはブラシがちょっと……。X68000を真似たATケースをプラ板で作ってしまいました。HPで写真載っけてます。

田崎晃(26)東京都豊島区 X68000XVI，AT互換機

ということで，今回の特別企画に近いことをやっていた方からのお便りでした。けど，Webページのほうはリンクぎれで紹介できませんでした。そうですね。プラ板という手もあったんですけどね。最初はフロントフェイスはスモーククリアレジンでとか，バックパネルはアクリル板でという話もあったのですが，結局，ほとんど総アルミになってしまいました。かろうじて，電源スイッチにバイオレットの集光アクリルが使われましたけど……。お仕事のほうはもう大丈夫なのでしょうか？　ペイントツールができたらぜひ送ってきてくださいね。

◆X68000や現在普通に使われているパソコン各種技術的な集大成という感じでとてもよい内容だと思う。おれは別に値段が高とは思わない。相応だと思いますよ。もちろん，安けりゃうれしいけど。個人的には信号処理関係の話題が出てくれるとうれしいです。

千代谷慶(24)東京都 G6-300

信号処理ですか。DSP関係もいろいろやっていきたいんですが，環境がいまひとつ特定できませんし。信号処理だけならとりあえずIPL関係が無難ですかね。

◆次々号の予定が21世紀というのはちょっと寂しい。Oh!Xの2000年問題か。すぐ役に立たないかもしれないが，いずれ必要になることもあるかもしれないので，このままのレベルでお願いします(もっと難しく専門的になってもOK)。

三枝史浩(31)兵庫県 X68000，AT互換機

ほらほら，それは1オリジンだからまだ20世紀だって。

◆なかなか寝つけないときにOh!Xを読むのが習慣となっている。とびきり難解な記事を熟読すれば，授業中に眠くなるのと同じ効果が得られるからだ(朝方なら放送大学という手もある)。睡眠導入にまで使えるのだからOh!Xじゃ実に懐が深い……。

中島民哉(29)埼玉県 X68000PRO

いえいえ，眠らせるようではまだまだ修行が足りないといったところでしょう。夜中に読むと目が冴えて眠れなくなるような本にしなくては。

◆昔ネット対戦ゲーム(X68000のジオグラフシール)にはまって以来，一度は自分で通信関連プログラムを組んでみたいともくろんでいます。DirectPlayを利用したチャットプログラムなどから……。ということで，VBのプレゼント企画期待しています。よろしく〜。

佐竹一生(34)福岡県 X68000ACE-HD

今月号の特集記事にもあるとおり，VB5CCEでもDirectXは使用できます。今号のCD-ROMに収録されたDirectX7.0a SDKさえあれば，あとはV5CCEを落としてくるだけで，通信対戦だろうが，ポリゴンバリバリだろうが大丈夫です。ではご健闘をお祈りいたします。

◆今回初めてはがきを出しました。しかし，いまになってプログラムの勉強をしておくんだったなと思い始めています。受け身ではなく，なにかを自分から提供できれば，本誌でもあった鉄の時代も変わってくるのかなと思います。話は変わりますが，いまのエミュレータ関係はすさまじいものがありますね。VRAM 1枚でかなりのものが動きますから驚きの連続です。そう考えるといまの日本のレベルって低いのかなと思ってしまう毎日です。いまの日本の若い方には頑張ってもらいたいですね。

菊池政行(28)福島県 AT互換機，X68030

なんのなんの28歳で年寄りじみたことをいっててはいけません。教訓：年齢は16進数で数えること。10代の若者はまだまだ学ぶべきです。人間50年としても，残りの一生を悔いたまますごすつもりですか？　もっと覇気を持っていきましょう。

◆確かに同人誌だ(笑)うん。

向井利宏(27)大阪府 AT互換機

まあ，確かに仕事のつもりでは作ってませんからねえ。

◆バックナンバーで入手しました。ジョイスティックをマウスにする方法をずっと探していました。記事を見つけて本当にうれしかったです。本来のX68000ユーザーではありませんが，今後とも細く長くよい記事の雑誌を出してください。特徴のないパソコン誌が出回りすぎてますから。　畠中規（？）神奈川県 PS/V改

復刊号のはがきでした。身体障害者用の機器を開発していらっしゃるようで，細かい動きの困難な方用の入力デバイスとしてお役に立つようであれば幸いです。

◆機種依存，言語依存，環境依存の話が多く，あまり面白みを感じられません。　高橋毅（28）神奈川県

ま，それは人それぞれでしょう。機種非依存の話ならともかく，言語非依存や環境非依存となるとプログラムの話はできなくなりますし。依存部の中からエッセンスを汲み取ってやってください。

◆付録のTREE構造がほしいなー（たとえCD-ROMでも）。そういえば，スタッフ募集って履歴書が必要なのではないのですか？

荻野潤（25）埼玉県 X68000，AT互換機

最初はつけてたんですが，なにかやるたびに作り直していると面倒なのでつけなくなりました。ドタバタしてて，復刊号のファイル一覧は実は正確ではなかったような。それとスタッフ募集（常時募集中）ですが，ライティングスタッフの募集が中心なので，あまり関係ありません。履歴書を送ってくださる方もいますが，要はなにがしたくてなにができるかってことだけですし。

◆もう少しで西暦2000年，本当の世紀末ですね。ちょうどいい時期なんで，そろそろ本当にパソコン買いますかね。といいつつ，ほかのものの後回しになってなかなかじゃないのが現実なんですけどね。

藤原彰人（29）岡山県

すでに2000年。1999年中に出すはずだったんですが，これがいろいろ事情がありまして。パソコンもなけりゃないで別に困らないんですが，置いといたほうがいいとは思いますよ。うちの場合は必要に迫られているってのがありますけど。

◆春号と夏号の間には出てないんですよね。6月中旬とか書いてあって，買い損ねたかなとか。第何号とか書いてください。買い損ねたらバックナンバーとかOKかな。　奥誠司（19）和歌山県

時期は不定期で出ております。ナンバーをシリアルに入れるというのは考えはしたのですが，もしかして定期刊になったら不自然だなあと時期別の名称にしました。いま考えるとまったく余計なことをしたものです。バックナンバーは小社販売局に問い合わせてください。たいていは書店で注文すればOKのはずですが。

◆私は絵心がまったくありませんので，「2D Laboratory」はまったくわかりません。でもゲームレビューが独特で面白いので，次はナムコゲームの新・旧作のレビューをお願いします。

真田知之（27）北海道 AT互換機

ま，絵心がなくてもいろいろ見て楽しむことはできますし，いろんなアルゴリズムを考えるとかいうのはできるものですよ。しかし，ナムコゲームですか……。いろいろ書きそうな人はいますけど，なんかいまさらな感じもしないではないですね。さてさて，どうしましょうかね。ご自分で書いて投稿するというのもアリなんですけど？

◆カラーページを減らして，値段を安くしてもらえませんか？　昔のOh!Xとまではいいませんが，2000円は超えないようにしてもらえると……。不景気でサイフがとっても軽いので。

小林憲司（31）愛知県

さて，きましたね。こういう要望をよくいただくのですが，はっきりいって，全部モノクロに変えても極端に安くなることはないと思いますよ。確かに，この本は開き直って作っているので，あちこち非常に豪華な作りになっています。昔のOh!Xはなにからなにまでキツキツに詰めて原価を下げていたのに比べると，編集ページは倍以上，広告はほぼなし，という状況ですから，付録ディスクつきの旧Oh!Xの2倍以上になるのは当然ですし，部数も昔ほどではありません（よく誤解している人がいるが，旧Oh!Xは部数だけならかなり出ていた。広告収入が低いのはいうまでもないけど）。ちょっと上乗せすればこれくらいの贅沢はできるわけです。まあ，普通の作り方にすると，400ページくらいになるのを押し込んでいるんで，実はかなり節約されているというのもあるんですけどね。で，なにより重要なのはほとんどがカラーだから成立している本でもあるという事実です。全部モノクロにするならともかく，ちょっと減らすくらいだと，逆に本が出なくなります。もしかして1色広告を出したいと思う人がいるかもしれないと，ちょっとだけモノクロページを作ってますけど，本当は全部カラーにしたほうが編集は楽なのです。ほとんどの記事がカラーページで見開き始まりの偶数ページで収まっているというのは決して偶然ではありません。この本のボトルネックは編集作業ですので，その辺に極力負担のない構成をとっているわけですね。前に書いたかもしれませんが，この本を作るに当たっては，ありとおらゆる反則技を駆使してますので。

◆Windows（マイクロソフト関連）の内容はいりません。ハード製作，およびでのデバイスドライバなどの記事が増えるとうれしいです。USBデバイス製作とか，PCIデジタル入出力ボードとか……。

堤博之（32）大阪府

BeOSの人ですか……。Beもフリーになって，少しは状況が変わるでしょうかねぇ。それはともかく，USBやPCI関連だと，インタフェイス部分だけで普通の人には手が出なくなりますからねぇ。なんとかしたいとは思っているのですが。

◆現在，会社にでC言語の研修を受けているところであり，Cによるプログラミングに興味があります。本誌にもプログラム関係の記事が多数載っていて勉強になります。

高尾浩（37）栃木県

X68030Compact，Presario9650

はい，どうもです。C言語がひととおりできるようになると，パソコンも活用範囲が全然違ってきますので頑張ってください。なにかまとまった作品ができたら，投稿のほうもよろしくお願いしますね。



▲夏倶秋水　広島県

このコーナーでは皆さまからのご意見を募集しております。綴じ込みのアンケートはがきや，e-mailなどでご意見，ご感想をお聞かせください。

e-mailでのあて先は，

aueki@softbank.co.jp

となります。

また，投稿イラストも随時募集しておりますので，

〒103-8501

東京都中央区箱崎町24-1

ソフトバクパブリッシング（株）

DOS/V magazine編集部内Oh!X宛

でお送りください。皆さまの参加をお待ちしております。

編集室から from E・D・I・T・O・R

SHIFT BREAK

▶はじめまして。水野と申します。Macintoshで，FutureBASICなどを使って細々とプログラミングをしている者です。このたび，古旗さんの助っ人で，AfterDarkモジュールの作り方について書かせていただきました。はなはだ未熟者ではございますが，以後，お見知りおきを。私の書いた記事が，皆さまのお役に立てれば幸いです。今回，作成したソースコード，および補足的な情報につきましては，私が作っているWebSite（http://www1.cds.ne.jp/~takaaki/）で公開しますので，ご覧ください。（水野）

▶なにをするにしても，周りから少し失笑を買ってしまうし，いつもどこかズレてしまうけれど，自分のやりたいことをするためには，カッコの悪さなんて気にしていられない。自分の気持ちに嘘はつきたくないから。多分こんな毎日こそが僕にとっては最高の時期。でも世間一般から見たら賢明な生き方とは決していえないのかもしれないけれどね……。
（井の中の蛙，されど空の高きを知る。飯田）

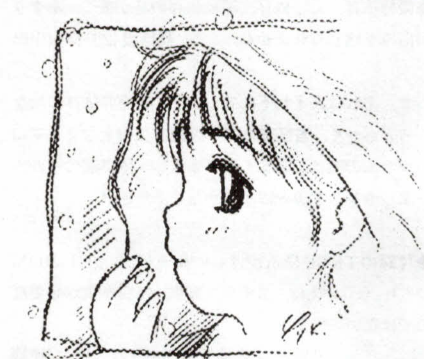
▶今回は，家庭的な雰囲気を作りたくてCGを描きました。でも，やはり家庭の温かさのような雰囲気を作るのは難しいですね。それから，現在はCGの制作環境をEx-toolsのShadeに完全に移行したので，まだ不慣れなところがあって，ちょっとぎこちなさを自分でも感じます。でも，CGを描き上げてみると，このソフトはとても手に馴染む感じがして心地よく感じました。これから使っていくことを楽しみにさせてくれるソフトですね。温かい雰囲気といえば，この間，谷山浩子さんのコンサートを見にいってきました。"幻想図書館「雪の女王」"というタイトルのもので，アンデルセンの雪の女王の童話を谷山さんの歌で語っていくという感じで，ちょっと普通のコンサートとは違う形式のものでした。歌声と曲がとても綺麗で，最後まで物語に釘づけになってしまうほど素晴らしいコンサートでした。このコンサートが終わったあとも余韻に浸りながら，こういうのを芸術というのかなぁってひとりで納得していました。ところで，私事で申しわけないのですが，私もホームページを開設してまして，最近，なんとかそれらしい形になってきました。3D CG関係の話題を中心にしていきたいと思っています。http://ha7.seikyou.ne.jp/home/tonta/ ですので，よかったら見にきていただけませんか。（田中）

▶今回は"コンピュータの登場"について記しましたが，コンピュータの登場自体よりも質と量がテーマとなってしまいました。ビデオゲーム登場から'80年代中盤までは物量の増加と同時に本質的な変化が大きかったのですが，近年では物量の増加へと変化しています。この号が世に出る頃にはPS2が発売されているのですが，本質的な面で新しいゲームがプレイヤーに新たな刺激を提供してくれることに期待しています。このまま容量勝負の作品が増え続けると，ヒット作の作者でさえ経済的に幸せになれない時代がやってきそうで，恐れるところが多いです。現在，新小岩にあるゲームセンターにてピンボールのリーグを実施しています。ピンボールの好きな方，好きだった方，そうでもない方。ぜひ，ご参加ください。次回は3月18日，19日になります。この号に掲載されているマシンもすべて設置されています！
http://www.mindware-corp.com/fot.html（市）

▶この本の分厚さに応えられるほど，コンピュータ的なものは描いていませんが，根底に流れている血潮は，私もきっと同じです。まだマイコンと呼ばれる時代から，ずっといままでこの流れとともに生きてきましたから。パーソナルにプログラミングするには，マシンの与えてくれる機能と性能が高すぎて，私には手が出せない時代ですけど，その代わりに4コマ漫画で，皆様のスキル向上を応援させていただきます。こんな小さな4コマでも，実はパソコンを使って必死に描いております。昔は，寝る前にベッドの上で，月に一度ハガキに4コマを描いていたことを思えば，時代は変わったなぁとしみじみ思う今日この頃です。4コマ1本にかかっている費用はいったいいくらに上がったのやら……。で，毎回，残念ながら掲載できなかった4コマは，私のHPで見られますので，よければ皆さま一度訪れてやってください（http://www.yo.rim.or.jp/~okamura/）。しかし，毎回もっとたくさん描こうと思っているのですが，思いどおりにいかないですねぇ……。（岡村）

▶DOGA-L3の開発，CGAコンテストの準備に忙しく，今号は原稿を書いてません。すんません。CGAコンテストのほうですが，今回の応募総数は406本にのぼりました。一昨年は128本，昨年は296本ですから，急激に増加しています。また，質の点でも，特に今年は従来の範疇を超えた逸品が揃い，単なるCG技術ではなく，映像作品としての内容を競う時代になったことを強く感じています。過去のCGAコンテストをご覧になってきた方も，そしてまだ見たことのない方も，今年は必ず"凄い！"，"面白い！"と感じていただけるでしょう。上映会には，ぜひご来場ください（詳しくは，イベントのコーナーにて）。（かまた）

▶10年におよぶ1K生活にピリオドを打つべく引っ越しを敢行したのはいいものの，かなり無茶なスケジュール＋インフルエンザでイラストの発注がきたときにはほとんど使いものにならない状態に……。久しぶりに思いっきり寝込んでしまいました。ということで今回1枚しか描いてません。すいません。でも唯一よかったのは寝込んで体重が3キロも減ったことかな（だからそういう痩せ方はやばいって）。（哲）



（Yoai）

▶それは昨年の夏の終わりのことだった。なんの気なしにカーテンをどけてみると，その向こう，AVラックの陰，畳の端から長さ6～7センチになろうかというキノコが4～5本密生して生えている！　うきゃあ！　有無をいわさず抜いて捨てたが，う～む，話にはよく聞くが，まさかこんなに簡単に生えるとは。確かに夏の間は窓を開けっ放しにして，ときおり雨が降り込んだりはしていたのだが。数日後，またなにかに引き寄せられるかのようにカーテンをどけてみると……いったん胞子が飛んでしまうと，絶滅させるのはなかなか困難らしい。いっそこのまま屋内プラントにしてしまおうか，2000年問題でライフラインが絶たれたときの非常食にでもしようかと思っていたのだが，それまではもたなかったようだ。結局なにもなかったし。あったことといえば，1月3日にTVアンテナが倒れたくらいか。それはそれで私にとってはライフラインなのだが。（i.k）

▶［映画］「本当に怖い」というウワサの「ブレア・ウィッチ・プロジェクト」見た。こんなものを金払ってみた自分が本当に怖くなった。え。低予算映画？　限度額あるだろ，低いほうにも。オイラも，紺のブレザーかなんか着込んでデジカム持って山に入って遭難したりして，しまいにゃ自分の野グソ姿撮影して一儲けしてやろうかしら。題名は「ブレザー・ウンチ・プロジェクト」。だめ？本当に怖いよ。[AGE OF EMPIREⅡ]　去年は「POPULOUS:THE BEGINNING」（Bullfrog）の攻略本を書いたが，今年は「AGE OF EMPIRES Ⅱ」の攻略本を書いた。今回の本は私が中心となりつつも，DOS/V magazineのスタッフ，某誌

のライターさんなどの協力の下に作られた本なので完成度は高い。AGEⅡの本はたくさん出ているがAGEⅡファンは絶対これを買うべし。ちなみに「火炎船」が「火災船」になっているのは私のせいじゃない。誤植だーっ。どんな船だーっ。さて，このAGEⅡ……「面白いのか，つまらないのか」非常に判断の困るゲームである。AGEⅡは，よほど実力に差がない限り逆転勝ちはありえないゲームなので，敗色が濃厚になると途端にやる気が失せてくる。一発逆転の要素がまったくなく，結局は物量戦に勝ったものが勝利する。これは実際の戦争もそうなので，戦略シミュレーションゲームという観点からすれば正統なのかもしれないが……。［PS2］PS2の発売を来週に控えた今日。やってないゲームが山積みなので発売日には手は出さない。にしても，ストリートファイターEX3はストⅡ時代のキャラクターがだいぶ復活しているので「あの頃」の感覚でプレイできそうでちょっとよさそう。プレイしたい。しかし，ディージェイはなぜいないのか。マシンガンアッパー打ちてぇ。さてさて，PS2はいったいどのようなかたちでネットワークに繋がるのかが楽しみ。ドリームキャストのようにモデムを内蔵しなかったからこそいろんな可能性が見えてくる（DCもモデムは取り外せるけど）。PS2，モデムカードだけ出ておしまい……なんていうつまらないことにはならないとは思うが……。Ethernetカードは最低でも出してもらわねばな。［Windows2000］Windows98から密かに削除されたMicrosoftFaxはWindows2000になっても復活しなかった。結構便利だったのになぁ。なぜなくしたの？　えっ？　バグが多すぎて担当者いなくなって，なくさざるをえなかったの？本当？　マイクロソフトさん，Windows2000にOffice2000ファミリーのFrontPage2000がインストールできません（実行ファイルだけコピーされない）。同じ2000がつく同士，普通動くと思うじゃないですか。え？　Windows98でインストールした状態でWindows2000にアップグレードすれば動きますよって？　それパッケージに書いておいてください。いまだWeb更新できず。［デジカメ］前号のOh!XでCCDの仕組み書いた関係で，ソフトバンクのデジカメムック「デジタルカメラLabs（仮称）」より執筆依頼がきた。私もデジカメの西川としてモデルとか撮影旅行したりして楽しい思いができるのかと思いきや，再びCCDの解説だった。そんでもって，富士フイルムの新世代CCD，「スーパーCCDハニカム」を見てきた。432万画素デジカメ「FinPix4700Z」はバーチャルピクセルで432万画素だった。そのあとエプソンが「HypictCCD」で484万画素のデジカメ「CP-900Z」だって。これも334万画素のCCDの映像をソフトウェア処理で484万画素の画像にする，まぁいってみりゃバーチャルピクセル。デジカメの画素数競争もちょっと各社壁を迎えてブレークスルーに悩んでいるみたいね。

（西川善司，http://www.z-z-z.gr.jp/zenji/）

▶本号は1999年11月発売予定だったので，改めて原稿を読み直すと非常に懐かしい感じがする。いまは1月末なわけで，その間いろいろなことがあった。ひときわ印象に残っているのがSPEEDの解散発表，ではなくて，石黒彩のモーニング娘。脱退，でもなく，昨年末の堀井雄二のドラクエⅦの開発凍結，休養宣言である。大人げないという誹りもあったが，周りが自分の考えをまったく理解していないことに対する怒りはもっとも。個人的には大いに共感できる。さすがに騒ぎは1日で収まったが，エニックスにとっては青天の霹靂だったろうなあと予想できる。まあ，開発中止にならなくてよかったよかった。さて，今号が出る頃には待望のPlayStation 2が発売されているはずだ。エモーションエンジンの歩留まりは改善したのだろうか。最初は0.25 μmでいくのだっけ。デジタル回路もサブミクロンルールになるとアナログ的要因が入ってくる。つまり，面積縮小，歩留まり改善，スピード向上を目指して0.18 μmにシュリンクすることにより，配線間隔が短くなりすぎ，配線間の負荷容量による遅延が，ゲート遅延を上回る恐れが生じる。潜在的に，シュリンクしても面積が小さくできない（原価低減できない），あるいは，動作スピードがかえって遅くなるという危惧があるのだ。製造元の東芝がこの問題をどうクリアしていくのかが見ものだ。Dolphinも期待していたのだが，ちょっとやばそう。Gekkoチップの製造元のIBMが高額なロイヤリティを要求するので，IBM以外に拡散ラインを確保できていないようだ。ここにきてチップ供給に不安が出てきた。そうえば，Dolphinは2000年冬発売予定が，知らないうちに2001年春に延期されていたな。もしかして，次世代ゲーム機の覇者はX-Boxだったりして（まさかね）。でも一番売れるのはゲームボーイアドバンスなのは間違いないような気がする。

（幸恵ちゃんに振り向いてもらいたいA.N.）

▶結局，今年度は1冊出しきれずに終わってしまった。1年間DOS/V magazineの副編集長をしてきたわけだが，当初予定だと年間半分以上いない副編というおかしな存在になるところだったのだから，少しはマシな状態になったのかもしれない。まあ，現状でも半分くらいはいたかなという程度でしかないのだけど。来期はまたちょっと違う仕事になるので，またまた予定が立たない。少なくとも2冊くらいは出したい……ところだけど。

（U）

古きよき20世紀の終わりに

意外と簡単にやってきた西暦2000年。PS2は発売され，PCのCPUも新世代へと移行しつつあります。21世紀まで万全の聴牌状態です。世間では古きよき時代の遺物と思われているらしいこの本ですが，基本的に21.5世紀を目指した方針でやっていることは変わりありません。内容が十分に伴っていないというのは課題なのですが。

本誌の内容について，編集側のなかではそれなりに一貫性があるのですが，あちこち必須パーツが抜けているので全体像がかなりぼやけてしまっていることは否めません。ビジュアル系の要素がさらに増えています。

「機種を問わない」

とはいえ，昨今ではx86系以外のCPUの話がこれだけ載っている本も珍しいでしょう。CPUがパソコンの中心であるという構図はまず揺るぎません。

最終的にはパソコンを，そしてOSを作るというところを目標としているのでCPUをはじめ，各種システムおよびハードウェアの基礎と応用は欠かせませんし，既存のシステムで学ぶべきところは学び，検討すべき点は検討するのも重要です。ハードウェア自体がプログラマブルであることは間違いありませんし，プログラマブルデバイスは今後の柱となる項目のひとつです。

どうやらPlayStation2ではPLDレベルでプログラミングが要求されているとも聞きますし（未確認で申しわけないが），時代はそういう方向に流れていると思ってもいいでしょう。

ハードウェアレベルのプログラミング，RISCプロセッサの理解，そして表現力。3Dグラフィックでは次世代の焦点は人体のモーションと衣服の表現となるでしょう。核となるべき物理演算の連載が進んでないのですけど，やがてはお洋服連載（今号には載ってないけど）がなんの違和感もなく受け入れられる世の中になることを信じましょう。

ということで，今号では，多数のコーナーが落ちました。次号以降ではちゃんと復帰する予定ですのでお楽しみに。次号の予定は一応6月頃ということにしておきますが，例によってあまりあてになりません。毎回発売日を延期しているようではいけないのはわかっているのですけど……。詳しい発売日などは，

http://www.softbank.co.jp/publishing/ohx/

をチェックするようにお願いします。

編集者は完全にパンク状態ですので，投稿その他，常時お待ちしております。

なお，今回は愛読者はがきの有効期限が短いので，はがきは早めに。期限切れの場合は切手を貼って送ってください。

microOdyssey

ゲーム関係の話をしていると，しばらく前に特許関係でもめていたコナミ，テクモ，ナムコといった一連の音ゲーの騒動が話題にのぼることが多い。結局，テクモはゲームから撤退したと聞く。ゲームのスタイルや処理方法についていちいち特許が取られるような世の中になってきている。

音楽演出型ゲームというべきジャンルが独創的なものかという点がひとつの問題なのだろう。被告になっている側は，Quest for Fameやパラッパが先だと主張していたみたいだが，Windows版でバーチャルピックを持って画面にあわせてギターを演奏していた奇特な人々はギタージャムを見てどうすればいいのだろうか。

そもそも音楽にあわせて画面を変化させるとかいうだけで特許になるくらいで，セガが昔それで痛い目にあっているのだがコナミは大丈夫なのだろうかと心配してしまう。

特許を取るのが悪いというわけではないが，どうもうさんくさいものが多いのだ。

たとえば，セガの公開特許申請の特開平07-227480，「ブロック連鎖テレビゲーム機」という発明を見ると，

【請求項1】複数種類のブロックがランダムに発生し移動して他のブロックと平面的に連結していくブロック連鎖テレビゲーム機において，検索基点のブロックを中心に周囲のブロックを一周に亘って検索して検索基点のブロックと同種類のブロックを検知する検索手段と，前記検索手段の検索により新たな同種類のブロックが検知されたとき検索基点を検知ブロックの位置に移動させる第1検索基点移動手段と，前記検索手段の一周に亘る検索により新たな同種類のブロックが検知されないとき検索基点を元のブロックの位置に移動する第2検索基点移動手段と，前記第1検索基点移動手段による検索基点の移動があったときのみ“1”を加算していき同種類のブロックの連鎖のブロック数を算出する計数手段と，前記計数手段により算出した同種類の連鎖するブロック数が所定数以上のときはその連鎖するブロックを消滅させるブロック消滅手段とを備えたことを特徴とするブロック連鎖テレビゲーム機。

ということで，落ちモノゲームすべてを囲い込むような方向で動いていることがわかる。平成6年の発明だが，泉大介のYACはどう処理してたっけ？　誰がやっても同じになりそうな気はするのだけど。

プログラムに関する著作権はコードそれ自体に関するものであって，方式ではないうえに，誰が書いても同じになるようなものは除外される。特許の場合は申請した者勝ちというわけだ。某氏の話だと，某ウィーゼル製作所では改行コードのCR＋LFとLFを判定するときに，後ろから調べる方法で特許を持っているそうだ。

こういった情報は公開されているので，特許庁ホームページの検索サービス，

http://www.jpo-miti.go.jp/ipdl/ipdl_a.htm

で確認できる。

調べると各社それぞれたくさんの特許を取っていることがわかる。セガの万歩計機能付きゲーム装置って奴もうさんくさいな。ハドソンが取るならともかく。

いろいろあるが，やや危険そうなものはコナミの「ビデオゲーム装置及びビデオゲームのプレイキャラクタ成長制御方法」だろうか。RPGなどにおける内部パラメータの変動の要因を戦闘とかではなく，メニューから選択したキャラクターの言動などの評価によって行うというものだ。

RPGや育てゲー，格闘モノなどへのバリエーションが記載されているが，基本的にエロゲー路線のノリだと思えば新鮮味はない。勝手にどれかのパラメータが増えるというのではなくてボーナス点をプレイヤーが任意のパラメータに割り振ることができるとかいう成長システムも提唱されているが，ちょっとそれは直接的にすぎはしないか？　これをより自然に実装するとスキルの修得とかアイテムの購入などという形態になるはずだ。プレイヤーの意思をキャラクターの成長方向に影響させる手段を持ったゲーム，という意味ならばそういう実装を行ったものは決して少なくないだろう。

特許申請を出しているにもかかわらず各社があまり主張しないのは，根拠が薄いということを各社理解しているからではないだろうか。下手に主張合戦になればつぶしあいになるだけだ。

そもそも，ソフトウェアというのはノウハウの共有で成り立っているものだ。常識範囲をわきまえずに独占権を与えるのは非常にまずい。また，誰しも当たり前のこととして処理しているものをいまさらながら「発明」されるのはなにか割り切れない。それって，無知な者ほど有利ではないのか？　ちゃんと審査されればいいのだが，通ったりするのは，やはり特許庁が無能だということだろう。

特許の異議申請にいちばんいいのは出版物だそうで，過去に昔のOh!Xとかが持ち込まれたこともあるようだ。これからも地道にやっていくしかないということか。（U）

次回予告

Oh!X 2000初夏号　2000年6月(？)発売予定

特集　携帯系プログラミング？

特別企画　オリジナルATを組み立てる第3弾

GALAXY KNIGHTSの制作（後編）

Oh!X 2000年秋号　2000年秋発売予定

特集　内容未定

Oh!X 2000春号
DOS/Vmagazine編集部　Oh!X制作実行委員会編

■2000年4月9日発行　定価2400円（2286円＋税）
■発行人　岡崎眞
■編集人　稲葉俊夫
■発行所　ソフトバンク パブリッシング株式会社
〒103-8501　東京都中央区日本橋箱崎町24-1
編集部　☎03-5642-8189　FAX 03-5642-3429
販売局　☎03-5642-8100
■デザイン　クニメディア(株)
■印刷　クニメディア(株)
(株)サンエー印刷

乱丁，落丁，CD-ROMの破損はお取り替えいたします。小社販売局までご連絡ください。
雑誌　65815-09

Oh!X 2000 春号

愛読者カード

郵便はがき

料金受取人払

日本橋局承認
3181

差出有効期間
平成12年5月
14日まで

103-8730

346

(受取人)
日本橋郵便局
私書箱358号

ソフトバンクパブリッシング(株)
DOS/V magazine 編集部 内
Oh!X 2000 春号 係行

フリガナ	年齢	性別
お名前	歳	男・女
ご住所 〒		
e-mailアドレス：　TEL URL：		
勤務先名 (学校名)　使用機種/OS	パソコン 使用歴　年	
購読パソコン誌		

Oh!X 2000 春号

愛読者カード

ENQUETE

Oh!X 2000 春号

読者アンケート

●本誌へのご意見，ご感想をなんでもお書きください。

●本誌の内容は？

□難しい　□ちょうどいい　□やさしい

●面白かった記事

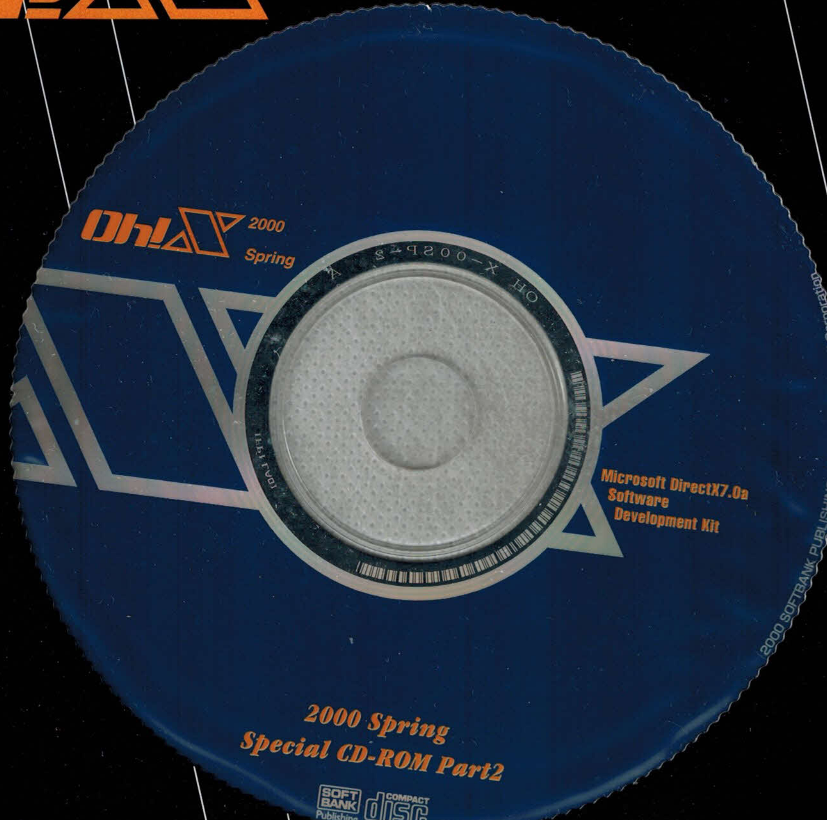
●面白くなかった記事

●興味を持っている話題をご記入ください。

DISC 1



Oh!X



DISC 2

2000春号特別付録 CD-ROM

Oh!X 2000 春号

収　録　内　容

DISC 1

シューティングゲームSION2000
NetBSD for X86k
MIPS Rシリーズ CPUシミュレータ
記事掲載各種サンプルコード

DISC 2

Microsoft DirectX7.0a Software Development Kit

ソフトバンクパブリッシング株式会社/出版事業部
DOS/V magazine編集部内Oh!X制作実行委員会

〒103-8501　東京都中央区日本橋箱崎町24-1
☎03-5642-8189（編集部直通）

制作協力　日立マクセル株式会社

SOFTBANK MOOK 2000春号
平成12年4月9日発行 ソフトバンク パブリッシング株式会社
定価 本体2,286円 +税

9784797310870

1929455022865

Printed in Japan クニメディア

ISBN4-7973-1087-1 C9455 ¥2286E

雑誌 65815-09